

DUTCHMAN LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

A Further Development of The
Massachusetts Institute of Technology
Computer Aided Design Executive System

by

LIEUTENANT COMMANDER DAVID LEE STONE, U.S. NAVY

B.S.E.E., Purdue University
(1973)

SUBMITTED TO THE DEPARTMENTS OF
OCEAN ENGINEERING AND MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF

OCEAN ENGINEER
and
MASTER OF SCIENCE IN MECHANICAL ENGINEERING
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1982

© David Lee Stone 1982

The author hereby grants to M.I.T. permission to reproduce
and to distribute copies of this thesis document in whole or

A FURTHER DEVELOPMENT OF THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
COMPUTER AIDED DESIGN EXECUTIVE SYSTEM

by

DAVID LEE STONE

Submitted to the Department of Ocean Engineering
on May 07, 1982 in partial fulfillment
of the requirements for the Degrees of
Ocean Engineer
and
Master of Science in Mechanical Engineering

ABSTRACT

The MIT Design Executive System, which is referred to as DEX, is a design oriented, structured interactive computer system. DEX provides the designer with a flexible but controlled environment in which to interact through DEX with program modules, databases, and the operating system. The structured design environment is provided through the use of menus.

A further development of the MIT Computer Aided Design EXecutive system provides truly dynamic loading of program modules at execution time. A dynamic storage capability is added along with the capability to determine file status and dynamically allocate files during execution of DEX. In the event of an abnormal termination of execution, this condition is trapped and any open database is saved.

Improvements are added to the present DEX database system, with compression of the database after delete. Backward pointers are added, and a proposal for a dynamically extendible database is provided.

Thesis Supervisor: Chryssostomos Chryssostomidis
Title: Associate Professor of Naval Architecture

ACKNOWLEDGEMENTS

To my wife Nancy whose enduring patience and encouragement has kept me going through it all. A special appreciation is extended to my children, Michelle, Daniel, John, and Edward who always sent me off in the morning with "we love you Daddy". Also to my thesis supervisor Professor Chrysostomos Chrysostomidis whose encouragement, thought provoking questions, and recommendations opened the way to the insight necessary to complete this project.

I would like to recognize the assistance provided by the staff of the Information Processing Center at the Massachusetts Institute of Technology. The assistance of many of these staff members in interpreting IBM documentation, and understanding system interaction is greatly appreciated.

I would like to extend a special note of appreciation to my good friend Bishop Melvin M. Scott, Jr. who has given me wise counsel and perspective, and who has often lifted burdens from my shoulders in time of crisis. Bishop Scott and his counselors Edward K. Abbott and Roger Brucks with whom I served in the Billerica Ward Bishopric of the Church of Jesus Christ of Latter Day Saints, have given me a great deal of support and strength in completing both my Church and Educational responsibilities. They are good friends whose support is greatly appreciated.

TABLE OF CONTENTS

Title Page	1
Abstract	2
Acknowledgements	3
Table of Contents	4
List of Figures	6
 1.0 Background and Thesis Organization	 7
1.1 Background.	7
1.2 Purpose	11
1.3 Preview of Things to Come	12
 2.0 The Design Executive System (DEX).	 14
2.1 The DEX System.	14
2.2 The Design Philosophy of the DEX System	17
2.3 The User in The Design Loop	18
2.4 Multiple Design Sequences	19
2.5 DEX Communicates in Plain English	21
2.6 DEX is Forgiving.	21
2.7 DEX has Many Sources/Destinations of Information	22
 3.0 The DEX Assembly Language Utilities.	 25
3.1 General Description	25
3.2 Dynamic File Management	27
3.3 Dynamic Loading and Unloading of Modules.	35
3.4 Dynamic Allocation of Storage	41
3.5 Error Recovery.	49
3.6 Other Utilities	51
 4.0 The DEX Database System	 56
4.1 General Description	56
4.1.1 The DEX Database Philosophy	57
4.2 The DEX Database Organization	65
4.2.1 The Physical Description of the DEX Database.	66
4.2.2 The DEX Hashing Function	71
4.2.3 An Example of Editing the DEX Database	71
4.3 Further Developments in the DEX Database	85
4.3.1 The Pointer Structure	88
4.3.2 The Modified Delete Structure	88
4.3.3 The Database Compression Feature (DBCMPR)	89
4.3.4 Dynamic Expansion of the Array Storage Area (DBX ECS)	90
4.4 Future Developments in the DEX Database	92
 5.0 Conclusions and Recommendations.	 95

References	98
Appendix A - Assembly Language Programming100
A-1.0 Why Assembly Language100
A-2.0 General Approach for preparing for a New Computer102
A-3.0 IBM/370 Specific Machine Structure106
A-3.1 Memory106
A-3.2 Registers108
A-3.3 Data109
A-3.4 Instruction Format113
A-3.5 Special Features117
A-4.0 Assembly Language Structure122
A-4.1 General Structure of Program125
A-4.2 Coding Conventions127
A-4.3 Subroutine Calling and Return Conventions128
A-4.4 Macro Organization133
A-4.5 Example Assembly Language Routine134
Appendix B - A Sample DEX Database Editing Session153
Appendix C - Assembly Language Program Listings157
Appendix D - Database Program Listings191

List of Figures

Figure 2.1 The DEX Menus	15
Figure 4.1 The Database SCHEMA, and Sample Data	67
Figure 4.2 The Physical Database Structure	69
Figure 4.3 Length is entered in the database	74
Figure 4.4 Diameter is entered in the database	75
Figure 4.5 Speedar is entered in the database	76
Figure 4.6 Height is deleted from the database	77
Figure 4.7 Array values are stored	78
Figure A-1 Load Assemble145
Figure A-2 Load Assemble with macro expansion147
Figure A-2 Excerpt from Load Listing150

CHAPTER 1

BACKGROUND AND THESIS ORGANIZATION

1.1 Background

In recent years the computer has come to the forefront of the design process, as numerous computer aided design systems have been developed. However, we as designers still do not realize the full potential of this powerful tool. There are several reasons for this. While the recent improvements in the user/ program interface have made the computer more straightforward to use, there are still several common problems which make the designer unwilling to accept the outlay of time required to change from one design program to another. Some of the common problems that these design programs, and the systems that contain them suffer from are:

1. Too often the programs are inflexible, and do not allow the user to deviate from the preconceived design process.

2. There is no consistent input/output procedure, and the output of one program cannot easily be used as input to another.
3. Because of the inconsistencies, the user must learn new procedures for each new program. This results in excessive training time requirements.
4. The program is often not forgiving, and after spending much time running the program an error can result in the loss of all output.
5. Inability of the programs to be transported to other facilities.
6. Often the programs are not user friendly. That is they were not designed with the ease of use as a major consideration in writing the program.

In 1974, researchers at the Massachusetts Institute of Technology and the University of Michigan joined together in a project to develop a structured computer aided design system that would eliminate or reduce the above characteristics. They named the system the Design EXecutive

System (DEX). The DEX system was written in a structured, top down manner, with each subroutine having a specific function. This was done to enhance transportability. If a given function was not available at a facility, then only the routine containing that function and the references to it would have to be changed. Also when transporting DEX, the few site dependent routines would be separate and could be easily changed without having to rewrite all of DEX. DEX can be adapted to almost any computer system that supports the Fortran programming language.

DEX provides an environment for running various application programs, called "modules". This provides ease of use and consistency by requiring all module programmers to interface with the user in the same manner. The DEX system was designed as a transactional manager that would insulate the user from both the module and the operating system, with DEX controlling all transactions. DEX consists of three levels or groups of programs:

1. DEX proper - The main body of DEX which provides the operating environment or "umbrella" of DEX, within which the user and the modules interact.

These routines provide the user with the consistent impression of the system.

2. EXTENDED DEX library - These are a collection of 45 utility subroutines and general functions which can be used by the module writer to facilitate the construction of the module. They are already set up to communicate with DEX properly. They include routines for reading, writing, editing data, unit conversion, and outputting messages and status indications.
3. Module - A module is a single subroutine or a complete set of subroutines written and executed together to perform a specific task. This is the actual application program which performs the calculations that the user is interested in.

The DEX system will be explained in more detail in Chapter 2. Celotto in his thesis [Celotto, 1981] gives the references for the earlier work on DEX, and gives an excellent example of how to write a module and how to run the DEX system. He also gives a detailed description of the Extended DEX library which was primarily his work.

1.2 Purpose

The purpose of this work is to make a further development in DEX proper. The first area of emphasis was in the DEX assembly language routines. While DEX is written primarily in Fortran for transportability, there are some system functions that were not available to DEX through Fortran at the Massachusetts Institute of Technology. It thus became necessary to write a few functions as site dependent IBM/370 assembly language routines. These functional areas were:

- User identification.
- System time.
- Enhanced file management routines.
- Enhanced dynamic loading and unloading of modules.
- Enhanced dynamic allocation of memory.
- Enhanced error recovery with recovery from abnormal termination.

This was my major area of emphasis.

The second area of emphasis was in the DEX database editing routines. To the existing database management system in DEX the following capabilities were added:

- Enhanced delete function.
- Bi-directional pointers.
- Compression of the database (elimination of deleted entries) when the database is full or at the users request.
- Expansion of the database array buffer when it is full through the use of dynamic memory allocation.

1.3 Preview of Things to Come

Chapter 2 provides further detail on the DEX system. Chapter 3 will look at the assembly language utility routines, and the DEX philosophy of file management, dynamic loading, and dynamic memory allocation. Chapter 4 will give

further detail on the DEX database system, what it was, what it is now, and how it should be developed in the future. Chapter 5 will give conclusions and recommendations for the future. Appendix A is an assembly language programmers guide, written for the user who desires to write or modify DEX assembly language routines. Appendix B is a sample session with DEX's database editor. Appendix C is the program listings for the assembly language routines addressed in Chapter 3. And finally, Appendix D is the program listings for the database routines discussed in Chapter 4.

CHAPTER 2

THE DESIGN EXECUTIVE SYSTEM (DEX)

2.1 The DEX System

The DEX System is a highly structured interactive Design Executive program which manages all interactions between: 1.) the user, 2.) application program "modules" operating within DEX, 3.) DEX databases in memory, 4.) other sources and destinations of data, and 5.) the operating system. The design philosophy of the DEX System is discussed in the next section. DEX provides a major control structure utilizing menus to communicate with the user. When in the DEX environment the user is prompted to enter an item from one of the DEX menus shown in figure 2.1 indicating the action that the user desires to be performed. The menu DEX.MAIN is the major DEX control menu, and contains the major DEX commands. DEX begins its interaction with the user by prompting for an item from this menu, and returns to this menu after the completion of the requested action or actions.

The menu DEX.DISP is reached by entering the menu item DISPLAY from the menu DEX.MAIN, and is used to display at

SENDER AN ITEM FROM MENU - DEX.MAIN
display menu all

\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	+	MENU	+	MENU	+	MENU	+	MENU	+	MENU	+	MENU	+
\$	+	DEX.DISP	+	DB-TYPES	+	DBEDCMDS	+	DEX.ALTR	+	DXYES-NO	+	DEX.MAIN	+
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	1	+	MENU	+	INTEGER	+	CREATE	+	TERSE	+	YES	+	LIBRARY
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	2	+	NEWS	+	REAL	+	STORE	+	VERBOSE	+	NO	+	HELP
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	3	+	MODE	+	ARRAY-RL	+	DELETE	+	KEYBOARD	+		+	DISPLAY
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	4	+		+		+	COMMENT	+	GRAPHIC	+		+	ALTER
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	5	+		+		+	EXPLAIN	+	ECHO-ON	+		+	TIDY
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	6	+		+		+	PRINT	+	ECHO-OFF	+		+	OPEN-DB
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	7	+		+		+	DUMP	+	DONE	+		+	EDIT-DB
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	8	+		+		+	SET-TITL	+		+		+	CLOSE-DB
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	9	+		+		+	GET-TITL	+		+		+	BEGIN
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	10	+		+		+	DONE	+		+		+	CONTINUE
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	11	+		+		+		+		+		+	SYSTEM
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+
\$	12	+		+		+		+		+		+	QUIT-DEX
\$	+	-----	+	-----	+	-----	+	-----	+	-----	+	-----	+

Figure 2.1 The DEX Menus

the terminal a menu or list of menus, DEX news, or the current mode. The system can be operating in either the DEX mode or a module mode. After the item is displayed control is automatically is returned to the menu DEX.MAIN. The menu DEX.ALTR is reached by entering the menu item ALTER from the menu DEX.MAIN, and is used to alter the state of DEX. An example might be to change from verbose to terse messages. This menu unlike DEX.DISP maintains control until the user has entered all desired menu items. In this case control is not returned to DEX.MAIN until the user enters the menu item DONE.

The menu DBEDCMDS contains the commands used to edit a DEX database. The menu DBEDCMDS is reached by entering the menu item EDIT-DB from the menu DEX.MAIN. The database edit commands will be discussed in Chapter 4. The menu DB-TYPES is used by the database edit routines to obtain the type of variable to be entered in the database. The menu DXYES-NO can be used by any routine to obtain a yes/no answer to a question.

The DEX.MAIN menu item LIBRARY displays a list of the modules currently in the DEX library. The menu item HELP displays a help file written by the module author to explain

the operation of a given module. The menu item TIDY is used to clear the graphics screen. The menu items OPEN-DB, EDIT-DB, and CLOSE-DB are used for manipulating databases, and will be discussed in Chapter 4. The menu item BEGIN is used to begin execution of a module. Upon completion of the module control is returned to the DEX mode. The menu item CONTINUE is used to resume execution of a module after execution of the module was interrupted by a user request to temporarily return to the DEX mode. The menu item SYSTEM is used to temporarily exit to the operating system, and the system command RETURN is used to return to DEX. Finally the menu item QUIT-DEX is used to terminate the DEX session.

2.2 The Design Philosophy of the DEX System

The design philosophy of the DEX system was to provide the user with the maximum flexibility, while still maintaining consistency through control of all interactions. Celotto in his work [Celotto, 1981] pointed out that "there are five characteristics of DEX which reflect the design philosophy of the system:

1. The user is in the design loop.

2. The system allows the design process to be executed in more than one sequence.
3. The system talks with the user in plain English.
4. The system is forgiving.
5. The system has multiple capabilities for input and output."

Let us explore each of these important characteristics.

2.3 The User In the Design Loop

The design process is iterative in nature, and there is not necessarily one correct flow through the procedure. Computer programs allow the relatively quick completion of complex and time consuming calculations, but often only in the sequence preconceived by the program writer. DEX allows the user to be in the design loop so the user can choose the sequence in which individual modules are executed. This is accomplished through dynamic loading and starting of modules. Additionally the user may choose to modify the

output of one module before it is input to another module or written to its destination. Thus the user can control the sequence of flow through the design spiral, and modify or insert additional data between steps.

2.4 Multiple Design Sequences

The flexibility offered by DEX is enhanced by the use of menus to allow the user to control the flow of execution. Thus the user has a wide choice of paths to follow through the design process. A menu is a list of options from which the user can choose the next step in the process, or the next variable to be defined. At present DEX allows a maximum of twelve items per menu, and a maximum of 25 menus. At any time the user can make a request to return to DEX from a module by typing the \$ sign followed by the name of the desired menu and menu item. Menu names and menu items can be up to eight characters long with no embedded blanks.

If the user is in the DEX environment, then DEX will prompt to enter an item from a DEX menu. If in the module environment, that is if the user has told DEX to begin a module, then the user will be prompted to enter an item from

one of the module menus depending on the point of execution of the module. If the user doesn't know the items in the menu, the menu items can be displayed by typing:

```
$ display menu menuname
```

Where \$ indicates a request to return to DEX, display is a menu item from menu DEX.MAIN, menu is a menu item from menu DEX.DISPLAY, and menuname is the name of the menu that you desire to be displayed. After reviewing the menu the user types:

```
continue
```

which is an item from menu DEX.MAIN, and the user is returned to the prompting message for the menu item. The menu items are of three different types:

1. Information - such as data needed for a module's execution
2. Commands - such as BEGIN for begin calculation, or DONE indicating the desire to terminate execution of this module.

3. Other Menus - such as INPUT implying proceed to the menu INPUT which might have as its items the possible sources of input.

2.5 DEX Communicates with the user in plain English

The messages and prompts which DEX gives the user are written in English sentences, and are designed to be as clear as possible. The responses which the user gives are items from the various DEX or Module menus. These menu items are also English words designed to be a logical response to the queries from DEX. Also the user can string together a series of menu items into a sentence which will cause DEX to match each word with the appropriate menu, resulting in a sequence of actions. Thus the interaction is very much like an oral dialogue. This makes the DEX system easier to learn as the dialogue is consistent.

2.6 DEX is Forgiving

Because of the complexity of DEX, the occurrence of errors is inevitable. Whether it be simply typing something

that DEX does not expect, or the attempt to load a program that does not exist. The designers of DEX tried to anticipate as many errors as possible. And where possible, diagnostic messages in plain English were included in the DEX message pool. Where possible, DEX advises the user of the error and then allows another try at the same place. If this is not possible DEX will return control to a previous routine or menu or back to DEX itself. In the event of a fatal error this investigator has added the capability to intercept the abort to the operating system, returning control to a DEX fatal error handling routine. This routine announces the problem to the user, saves any open database, and gives the user the option of terminating execution (recommended since the status of the system is unknown), or continuing execution until things have deteriorated beyond recovery.

2.7 DEX has Many Sources/Destinations for Information

In Celotto's work [Celotto, 1981], he used the term "information" rather than input or output, because the terms "input", and "output" are too limiting in concept. He rather talked of "sources" of information, and "destinations" of

information. This writer will adopt the same convention. DEX enables the communication of information by the dynamic allocation of databases and files. DEX distinguishes between two types of files, a.) databases, and b.) disk files. In addition to these two types of files, DEX can communicate information to and from the terminal (or plotter) in the form of alphanumeric characters or graphics. The terminal is the destination for DEX messages, and the source for menu entries by the user. The DEX operating environment can be seen to have five sources of information and four destinations; they are:

1. DEX - created and edited databases which can be saved on or loaded from disk.
2. The user at the terminal using DEX utility routines to read or write alphanumerics.
3. The user at a graphics device using DEX graphics routines to read or create x-y co-ordinate plots, or other graphical data. (This capability has not yet been implemented in the present version of DEX at MIT.)

4. Sequential disk files.

5. Module default data (source of information only).

The reader is referred to the work by Celotto[[Celotto, 1981] for a more detailed description of how to run DEX, and of the Extended DEX library of utility routines. A description of the DEX database system will be given in Chapter 4.

CHAPTER 3

THE DEX ASSEMBLY LANGUAGE UTILITIES

3.1 General Description

As previously stated the philosophy of DEX is to provide the most flexibility to the designer using DEX, while still maintaining uniformity of dialogue and consistent information transfer. As a result DEX is a complex but structured program. Since the user is in the design loop, DEX has no way of knowing in advance the path that the user will follow through the design sequence. For this reason it cannot be determined in advance which files the user will need access to, or which modules the user will need loaded. Additionally the user may start using a database which might become full, and need to be expanded.

As a result, if all of the possible combinations were attempted to be provided for when DEX was compiled, the size and cost of DEX would be prohibitive. It would not be feasible to attempt such an undertaking, even if the resources were available. For this reason there are several functions which are essential to the dynamic nature of DEX. They are:

1. Dynamic file management.
2. Dynamic loading and unloading of modules.
3. Dynamic allocation of storage.
4. Error recovery.

Additionally there are several cosmetic or accounting capabilities that would enhance the DEX environment. These are:

1. Obtaining time from the system,
2. The obtaining of a users ID for communication and accounting of DEX use.
3. The ability to interrupt DEX, go to the system to perform some function, and then return to DEX at the point where you stopped.
4. The ability to shift the contents of a memory location either left or right.

However, these types of functions are generally not available to the IBM/370 Fortran programmer, and therefore it was necessary to develop them in assembly language at MIT. The description of each of these capabilities are given in the subsequent sections. The code listings for the assembly language routines are included in Appendix C.

3.2 Dynamic File Management

In the traditional program organization the input/output files must be allocated to the logical unit number either before the program is executed or in an "open" statement in the program. This results in a fixed file structure, with the unit number being allocated to one file and one file only for the entire execution of the program. In the DEX System, it is unknown which modules the user will invoke, and therefore what files will need to be allocated to which units. DEX requires the dynamic allocation of files at execution time, in order to provide the capability to change file allocation at any time. Also it is necessary for DEX to be able to check the existence of a file before attempting to allocate it. DEX needs to be able to CLOSE files after use and free the allocation, so that the unit can be

used for other files. Before allocating a file DEX must be able to determine if a file is already attached to the desired unit, and if so what file. The following is a list of the assembly language routines used for dynamic file management:

- **ALLOC** - Allocate a file for read or write. DEX allocates logical unit number (LUN) 18 as output device (OUTDEV), and LUN 19 as input device (INPDEV). Other assignments of LUN used by DEX are:

Messages device	MSGSDV=13
Usage device	USEDEV=14
Information device	INFODV=17
Database device	DBSDEV=12
News device	NEWSDV=15
Help device	HELPDV=16

- **CKFILE** - Check for the existence on the disk of a given file by its name.
- **CLOSE** - Close an open file.
- **FREE** - Free an allocated file.

- `QUERY` - Query what file if any is allocated to a given logical unit number (LUN).

The routines `ALLOC` and `CKFILE` were existing routines modified by this investigator. The routines `CLOSE` and `FREE` were existing routines which were not modified. The `QUERY` routine is a new routine developed by this investigator. The description of these routines is given in more detail below.

`ALLOC` The routine `ALLOC` is a combination of the previous routines `ALLOCW` and `ALLOC`. It is used to allocate a fortran I/O file to a given unit number. A third argument has been added, which is a `WFLAG` (write flag) which if `.TRUE.` tells the routine to allocate the file for write with the pointer at the end of the file. `ALLOC` is an Integer Function. The calling sequence is,

```
RET = ALLOC(LUN,FILENAME,WFLAG)
```

Where, `RET` is assigned the integer value zero if the file was successfully allocated, and a non zero return code if the file could not be allocated. The current version of the MIT IBM operating system returns the

value 24 for all the error conditions, so that they cannot be distinguished.

LUN is the Logical Unit Number which is the unit to which the module or DEX is writing or reading from.

FILENAME is a 24 byte string containing the CMS name of the file, segregated into 8 byte pieces. If the filename is given as a '*', then the terminal is allocated.

WFLAG is a flag which indicates that the file is to be allocated for write when it is .TRUE..

CKFILE The routine CKFILE is a logical function which checks the existence of a given file. This routine uses the FSSTATE macro which is a Conversational Monitor System (CMS) macro which checks the existence of a file. Error handling was added so the routine returns an RCODE to indicate the reason for a failure to find the given file. The calling sequence is,

```
RET = CKFILE(FILENAME,RCODE)
```

Where FILENAME is an 18 character string containing the full filename.

RET is a logical variable which is set .TRUE. if the file exists, and .FALSE. if the file does not exist.

RCODE is an error return code of integer type. The meaning of the return codes are,

RCODE = 0 No error

RCODE = 1 Invalid character in FILEID

RCODE = 2 Invalid FILEMODE

RCODE = 3 File not found

RCODE = 4 Disk not accessed

CLOSE The CLOSE routine was not modified by this investigator, but is included here for completeness. This routine closes a file for fortran to allow for file cleanup after use. The calling sequence is,

CALL CLOSE(LUN,&NOTGOOD)

Where LUN is the Logical Unit Number to which the file to be closed is attached (the LUN is in full word binary)

&NOTGOOD is for a fortran RETURN I convention. It is the label number of the line in the fortran routine which called CLOSE where control will be returned if the CLOSE is not successful. This is accomplished in assembly language by returning a 0 in register 15 if the normal return is to be taken, and a 4 in register 15 if the &NOTGOOD return is to be taken. For example;

```
CALL CLOSE(18,&33333)
RCODE = 0
go to 99999
C...    The file could not be closed
33333 RCODE = 1
      .
      .
99999 RETURN
      end
```

FREE The FREE routine was not modified by this investigator, but is included here for completeness. This routine frees

the file from the given LUN. The present version ignores the filename and frees whatever file is attached to that LUN. The calling sequence is,

```
CALL FREE(FILENUM<,FILENAME>,&NOTGOOD)
```

Where FILENUM is the LUN in character format

FILENAME is the name of the file to be freed; it is currently ignored.

&NOTGOOD is for the fortran RETURN I convention, and is as described in the CLOSE routine description above.

QUERY The QUERY routine was developed by this investigator. The QUERY routine makes use of the RDJFCB (Read Job File Control Block) system macro, and the IBM CMS (Conversational Monitor System) macros CMSCB and DEVTYPE [IBM, 3,4,5]. When passed a fortran I/O unit number (UNIT), this routine reads the system Job File Control Block (JFCB) for FTNNF001, where NN = unit number. The JFCB contains information about any file which is attached to that unit number. The CMSCB macro

(CMS Control Block) is then used to map the JFCB. The CMSCB macro is a series of labeled fields with the same length as the fields in the File Control Block. These labels can then be used to extract data from the JFCB. The FCBREAD field is checked to see if a file was attached. If none is then a .FALSE. is returned. If a file is attached a .TRUE. is returned, and the FILENAME, FILETYPE, and FILEMODE are extracted from the JFCB and returned. The CMS DEVTYPE macro is invoked to determine the device type, and this is returned. The calling sequence is,

```
TRUFLG = QUERY(UNIT,RCODE,NAME,TYPE,MODE,DEV)
```

Where UNIT is the fortran LUN for which information is desired.

RCODE is a return code which is not currently used. It was included as an argument in the event that later changes in the system control block structure necessitated a return code to identify the reason for failure of the query.

NAME is the filename returned.

TYPE is the filetype returned.

MODE is the filemode returned.

DEV is the device to which the unit is attached and is returned to the calling routine as an 8 character device. The possible device returns are,

PRINTER
READER
TERMINAL
TAPE
DISK
PUNCH
CRT

If no file is attached to the UNIT then NAME,TYPE,MODE, and DEV are set to zero.

3.3 Dynamic Loading and Unloading of Modules

The dynamic loading and unloading of modules is an important feature of DEX, and is an integral part of

the system, providing much of DEX's flexibility. In the previous version of DEX at MIT dynamic unloading was not implemented, and dynamic loading was simulated by reserving a very large storage area at the time DEX was initialized. Then when a module was loaded it would always be loaded at the start of this area. This allowed modules to be dynamically loaded, but caused several problems. First it was necessary to waste storage, because the area had to be sufficiently large to hold the largest routine in the DEX library. Additionally if two modules loaded were of different size and happened to use the same name for some of their routines, linkage conflicts could arise. This investigator developed the following assembly language routines to provide truly dynamic loading and unloading of modules. These routines make use of the OS/VS1 supervisor services macros. [IBM,6,7,8]

- LOAD - Dynamically load a module into memory and return the entry point.
- START - Dynamically start a loaded module.
- UNLOAD - Unload a previously loaded module.

These routines will now be described in greater detail.

LOAD The LOAD routine loads a module into main storage and returns the entry point address. It uses the OS/VS1 Supervisor 'Load' macro [IBM,7]. LOAD assumes the filetype is TEXT. LOAD calls the routine CKFILE with the filename to be loaded and a filetype of TEXT, to check that the file to be loaded exists. LOAD is a logical function, and if the load is unsuccessful a .FALSE. is returned. If the module is loaded, it's entry point is returned and LOAD is .TRUE.. This routine is used in Appendix A as an example. The calling sequence is,

```
TRUVAL = LOAD(FILENAME,ENTRY,RFCODE)
```

Where FILENAME is an 8 character filename, and a filetype of text is assumed.

ENTRY is the returned entry point address where the module was loaded.

RFCODE is a fatal return code which is set to one prior to entry. If no fatal error occurs the RFCODE is reset to zero. However, if a

fatal error occurs then execution terminates abnormally and RFCODE is still equal to one. This indicates to the DXABND routine that the error occurred in LOAD.

The concept of a fatal return code introduced above, was introduced by this investigator to identify the routine in which a fatal error occurred. The fatal return code (RFCODE) is passed in common to the DEX abnormal end routine (DXABND). The routine DXABND will only receive control if an abnormal end has occurred. The value of the RFCODE when DXABND receives control indicates the routine where the error occurred. Prior to calling any routine where a fatal error could occur RFCODE is set to the value assigned to that routine. Thus if a fatal error occurs control will not return normally but will go to DXABND with the RFCODE still set. If the routine terminates normally then the RFCODE can be reset to zero. Currently only four RFCODES have been assigned. It was decided that further assignments of RFCODES could be made as experimentation with DEX indicated the need. The current assignments are:

RFCODE = 0 No Fatal Error

RFCODE = 1 Logical Function LOAD

RFCODE = 2 Subroutine START

RFCODE = 3 Logical Function FREEMN

RFCODE = 4 Module program

START The START routine when passed a module entry point (which was obtained using LOAD) passes control to that routine, and then returns control to the calling routine after the module has completed. In other words, it provides linkage between DEX and the module program. The calling sequence is,

CALL START(ENTRY,RFCODE)

Where ENTRY is the entry address for a module as obtained from the LOAD routine.

RFCODE is a fatal return code which is set to two prior to calling this routine. If no fatal error occurs then RFCODE is set to zero before returning to the calling routine. If a fatal

error occurs then execution is terminated abnormally, and the RFCODE is still equal to two. This then indicates to the DXABND routine that the error occurred in START.

UNLOAD The UNLOAD routine deletes a previously loaded module from main storage. The UNLOAD routine is a logical function and returns UNLOAD = .TRUE. if the unload is successful. If the indicated module can't be found then UNLOAD is returned as a .FALSE.. The UNLOAD routine uses the OS/VS1 Supervisor 'DELETE' macro [IBM,7]. The calling sequence is,

TRUVAL = UNLOAD(FILENAME)

Where FILENAME is an 8 character name of a module loaded using the LOAD routine.

No fatal return is necessary as there is no fatal error condition.

3.4 Dynamic Allocation of Storage

The traditional storage approach is that an array must be dimensioned statically to the worst case size. This is not only inflexible, but is wasteful of storage. In the DEX System the dynamic allocation of storage is essential to the very philosophy of the database, and it is expected that the allocation of storage should expand and contract with the data stored. This is important if DEX is to be a truly flexible design tool.

- FREEMN - Free previously allocated storage.
- GETMN - Get a block of main storage and return its entry point.
- READEC Read a block of memory at a location offset from the beginning of an existing memory area.
- WRITEC Write a block of data to memory at a location offset from the beginning of the memory area.

The routines FREEMN, and GETMN were developed by this investigator to provide the dynamic storage allocation capability in the MIT version of DEX in a manner that would facilitate the future development of a dynamic database. The previous version was locked at 2000 words of main storage for the database and there was no capability to free the storage and get an enlarged storage area. The routines READEC and WRITEC were existing routines based on the previous implementation which simulated the Control Data Corporation (CDC) ECS storage scheme, but with a fixed amount of storage. The basic structure is still the same, but in the current version the READEC and WRITEC routines were modified to take the pointer and length information describing the storage area as arguments. The routine GETMN is called to get a block of storage of any size up to the maximum size of 16 million bytes. However, the system has imposed a currently authorized limit of 1 million bytes on the DEX account. This limit can be raised but as it is raised the system is slowed, and limits are placed on the use of DEX. The pointer to the storage area is returned and placed in common. This information is then used to manage the storage area, allowing a new area to be obtained, the old copied, and then the old released using the FREEMN routine.

FREEMN The routine FREEMN frees a block of storage via the system FREEMAIN macro. The number of words of storage and the pointer to the area to be freed are passed as arguments. The calling sequence is,

```
FREFLG = FREEMN(NWORDS,POINTR,RFCODE)
```

Where NWORDS is the number of words of storage to be freed.

POINTR is an integer pointer to the start of the main storage area to be freed.

RFCODE is a fatal return code which is set to three prior to entry. If no fatal error occurs the RFCODE is reset to zero. However, if a fatal error occurs then execution terminates abnormally and RFCODE is still equal to three. This indicates to the DXABND routine that the error occurred in FREEMN.

FREFLG will be .TRUE. if the area is successfully freed. The only reason that it would fail is if the size exceeds the allowed storage

size. This would cause an abnormal end which is trapped for.

GETMN The routine GETMN acquires a block of storage via the system GETMAIN macro. The number of words of storage to get is passed as an argument. A pointer to the area is returned. The virtual storage area obtained by the GETMAIN macro begins on a doubleword boundary (see Appendix A-3.1), or a page boundary. The area is not cleared to zero when it is allocated. This routine does not need a fatal RFCODE because the GETMAIN macro is invoked with a conditional code. If the storage size requested exceeds the allowed limit then the GETMAIN is ignored, and GETMN is returned as a .FALSE.. The calling sequence is,

GETFLG = GETMN(NWORDS,POINTR)

Where NWORDS is the number of words of storage to be acquired.

POINTR is an integer pointer to the start of the main storage area.

READEC The routine READEC reads a block of storage from the simulated CDC Extended Core Storage (ECS) area obtained by the GETMN routine. The block of storage is read into a string (symbolic name) starting at the first byte. The block begins at a given offset from the start of the ECS area. This routine is used to read a stored array or comment from the ECS area. Comments are known to be of fixed length 64. Arrays are stored with the length of the array in the first location and the n elements of the array in the next n locations. The calling sequence is,

CALL READEC (STRING, OFFSET, LENWDS, ECSPTR, ECSLEN)

Where STRING is the symbolic name of the array where the words of data are stored as they are read.

OFFSET is the number of words that the desired block is offset from the beginning of the ECS area pointed to by ECSPTR.

LENWDS is the number of words of storage to be read (length in words).

ECSPTR is the pointer to the beginning of the ECS area. ECSPTR is returned by GETMN when it acquires the ECS area.

ECSLEN is the length of the ECS area and is used to check that the request is not an attempt to access beyond the area. If it is the request is ignored and a return code of 4 is returned in register zero. This check is included to make the routine general, but is presently redundant as DEX makes a check prior to calling READEC. The check is made in the routine DBAPTR that the NEXECS plus LENWDS does not exceed MAXECS. If it does DBAPTR is set .FALSE., and upon return to DBEDIT, the routine DBXECS is called to expand the ECS area.

The READEC routine reads a block of storage LENWDS long starting at ECSPTR plus OFFSET into STRING. The sequence of action when using READEC to read a database array from main storage would be; first the pointer to the array relative to the start of ECS would be obtained from the database. Then READEC would be called with that offset and a length of one. The value returned would be the length of the stored array.

READEC would again be called, but with an array as the string location, OFFSET+1, and the length of the array just read with the previous READEC call.

WRITEC The routine WRITEC writes a block of storage into the 'ECS' storage area obtained by the GETMN routine. A block of storage is taken from string and stored in the ECS area. The block is stored at the given offset from the start of the ECS area. This routine is used to write an array or comment into the ECS area. Comments are known to be of fixed length 64. Arrays are stored with the length of the array in the first location and the n elements of the array in the next n locations. The calling sequence is,

CALL WRITEC (STRING, OFFSET, LENWDS, ECSPTR, ECSLEN)

Where STRING is the symbolic name of the array whose contents are to be moved into the ECS area.

OFFSET is the number of words that the block is to be offset from the location pointed to by ECSPTR.

LENWDS is the number of words of storage to be written (length in words).

ECSPTR is the pointer to the beginning of the ECS area. ECSPTR is returned by GETMN when it acquires the ECS area.

ECSLEN is the length of the ECS area and is used to check that the request is not an attempt to access beyond the area. It is handled in the same manner as in READEC.

The WRITEC routine writes a block of storage LENWDS long from STRING into the ECS area starting at ECSPTR plus OFFSET. The sequence of action when using WRITEC to write a database array into main storage would be: First the next available ECS location would be determined from the variable NEXECS, then WRITEC would be called to write the length into the next available location. WRITEC would again be called to write a pointer to the previous block into the next location. A pointer back to the database entry would be written, and then WRITEC would be called with the array and LENWDS equal to the length of the array.

3.5 Error Recovery

As has been previously discussed, DEX is a very forgiving system. If the user enters a number in the wrong format, then DEX announces it and indicates which data items must be reentered. If the user is prompted for a menu item, and enters an incorrect response, DEX again indicates the error and allows the user to recover. However, there are some system functions which are not forgiving, and cause an abnormal termination, such as an error in a module which tries an illegal action. As a result a routine was developed to trap the abnormal end condition in the system and reroute it to a DEX fatal error handling routine.

- ABTRAP - Intercepts abnormal termination, and transfers control to the fortran routine DXABND (developed by this investigator). The routine DXABND announces to the user that an abnormal termination has occurred, and gives the user the option of saving any open database. It then gives the user the option of continuing or terminating the session.

The routine ABTRAP uses the CMS macro HNDSVC to set up a trap for a supervisor call SVC 13 which is the command used by the CMS operating system to branch to the system abend routine. The HNDSVC macro sets up the trap with an address where control is to be transferred anytime the SVC 13 command is issued. The address used for the trap is the label ABNORM in the ABTRAP routine. After the trap has been successfully set ABTRAP returns control to DEX and indicates successful completion by returning with ABTRAP .TRUE.. Then if at anytime later an abnormal condition arises, the operating system takes control and issues an SVC 13 to call the system abend routine. This condition is intercepted by the trap and rerouted to the location ABNORM in the routine ABTRAP. This section of assembly language code clears the trap to prevent a recursive abend sequence, and branches to the DEX routine DXABND. DXABND is a fortran routine which informs the user of the abnormal end condition, checks the RFCODE to determine which routine caused the error, provides the opportunity to save any open database, and then gives the user the option of terminating or continuing the DEX session.

The DXABND routine makes the least number of calls possible because of the uncertain status of DEX and the operating

system. The primary objective here was to save the database so that all would not be lost. However, if the user desires an attempt will be made to recover and continue operating. This action is not recommended because the user's virtual machine has been altered and the conditions will continue to deteriorate. The calling sequence is,

```
TRUFLG = ABTRAP(DUMMY)
```

DUMMY is a dummy argument to satisfy local fortran logical function convention that a function requires an argument.

3.6 Other Utility Routines

It would be beneficial for DEX to have several other capabilities. It would be useful to be able to identify the user's logon ID in order to keep an account of DEX users, and also to include the name in the welcome message.

Often it is convenient, when running a program, to return to the system to check something, or perform

some system function and then return to the program to resume execution.

Because of the large range of possible functions which could be performed under DEX, it would be useful to have the date and time indicated on the record of each DEX session so that the chronology of the design could be kept.

In assembly language coding it is sometimes necessary to convert from words of memory to bytes. This can be done by shifting the contents of the word to the left. Also some system macros return the results packed into a single register with an address portion and perhaps a length portion. This information can be more easily extracted by shifting the contents of the register to purge the undesired portion. Thus a routine which shifts a word of memory a given number of bits would be useful. The assembly language routines which provide these capabilities are:

- ISHIFT - To shift a word of memory left or right a given number of bits.

- JGV MID - To obtain the users Logon ID from the system.
- SYSTEM - To exit to the system and then be able to return to DEX after completing the desired function. It should be noted that ONLY transient system commands, and commands which are nucleus-resident are possible. They are:

The nucleus-resident CMS commands,

CP	GENMOD	START
DEBUG	INCLUDE	STATE
ERASE	LOAD	STATEW
FETCH	LOADMOD	

The transient system CMS commands,

ACCESS	HELP	RELEASE
ASSGN	LISTFILE	RENAME

COMPARE	MODMAP	SET
DISK	OPTION	SVCTRACE
DLBL	PRINT	SYNONYM
FILEDEF	PUNCH	TAPE
GENDIRT	QUERY	TYPE
GLOBAL	READCARD	

- TIME - To obtain the time of day and date from the system.

These routines were developed by Jeff Rice, except for JGVMID which was developed by J. Graham of the MIT Information Processing Service and are presented here for completeness. The calling sequence is,

```
CALL ISHIFT(WORD,NBITS)
```

Where WORD is the word to be shifted

NBITS is the number of bits to shift. If it is positive the shift is to the left if negative the shift is to the right.

CALL JGVMID(VMID)

VMID is the user ID who is currently logged on and using this program. This is the return argument from the system, and is 8 bytes in length.

CALL SYSTEM _ Branches to the CMS subset.

CALL TIME(String)

Where String is the symbolic name where the time and date are to be stored. The format is HH.MM.SS.TH YY/DDD. This routine uses the supervisor services macro TIME to get the system time, and then extracts the time/date information in the correct format.

CHAPTER 4

THE DEX DATABASE SYSTEM

4.1 General Description

The DEX Database System identifies entries in the database using the name of the variable or datum as a key. The variable's location in the database is not important. When the variable is placed in the database or later needs to be located for data manipulation, its address is calculated using a "hashing" algorithm. The hashing algorithm translates the name (sometimes called a key) into a number which is uniformly scattered or randomized. This number is then used to determine where the element is to be stored [Donovan, 1972; Martin, 1977]. This technique and the logical and physical database structure will be explained in section 4.2. The formal description of the overall database structure is called a "schema" [Martin, 1977].

4.1.1 The DEX Database Philosophy

The DEX Database System was designed to provide the user with the maximum flexibility in database manipulation. A new database can be constructed and used, or an existing database can be loaded and used, by either the module program, or the user using database manipulation menu items from the menus DEX.MAIN or DBEDCMDS. Opening a database means that it is opened in memory for manipulation. If it is a new database, then memory is initialized for it. If it is an existing database which is stored on disk, it will be loaded into memory if necessary. No database will be loaded if a copy of the database requested already exists in memory. After the database has been created and used, it will not necessarily be saved on disk. DEX will prompt the user to determine if the user desires that the database be saved. Databases can be manipulated by either a module program or the user in the DEX environment. The possible methods are:

1. A database can be manipulated from within a module program using the database management routines directly, or using the Extended DEX Library. The

Extended DEX Library was created to facilitate the module author in handling different sources and destinations of information including databases. The reader is referred to the work by Celotto [Celotto, 1981] which gives a description of the Extended DEX Library routines, and how to use them. The basic DEX database management routines are:

DBOPEN - Opens a database in memory. If the database is new memory is initialized for the new database. If the database already exists on disk it will be loaded into memory and used. DEX will not load a database if it is already in memory.

DBVINS - Insert a new variable name (key) into an open database in memory.

AGET - Retrieve real array values from the database in memory, and store them in the indicated array. (in all of the references to the database below the database is in memory)

IGET - Retrieve an integer value from the database for the given key, and store it in the indicated integer variable.

CMTGET - Retrieve the comment associated with a given key, and store it in the indicated variable. The length of the comment can be up to 64 characters.

RGET - Retrieve a real value from the database for the given key, and store it in the indicated real variable.

TGET - Retrieve the title of the open database, and store it in the indicated variable. The title can be up to 64 characters long.

APUT - Store the values of an array in the database

CMTPUT - Store the 64 character comment describing the datum (key).

IPUT - Store an integer value in the database for the given key.

RPUT - Store a real value in the database for the given key.

TPUT - Store a 64 character string into the database as its title.

DBVDEL - Removes a variable name(key) from an open database by making the NODE where it is stored unavailable to the user. The NODE is flagged for later compression of the database.

DBCLOS - Save the database on disk if the user desires, and close the database.

2. The user can manipulate a database, while in the DEX mode, using database commands from the menus DEX.MAIN and DBEDCMDS. The Database Edit Commands menu (DBEDCMDS) is reached by entering the DEX.MAIN menu item EDIT-DB. The menu items on the menu DEX.MAIN which involve databases, and the routine called are:

OPEN-DB - calls DBOPUT - The DB Open Utility routine checks to see if there is already an open database. If there is the user is given the option of saving it. If the user wants to use a different database a name is obtained, and the user is given the option of using a saved database. If a saved database is to be used it is loaded into memory. If a new database is to be created the database memory area is initialized.

EDIT-DB - calls DBEDIT - The routine DBEDIT prompts the user to enter an item from the menu DBEDCMDS. The Database Edit Commands are given below.

CLOSE-DB - calls DBCLUT - The DB Close Utility routine announces the name of the open database if one exists, and gives the user the option of saving the database with the same or a new name. If the database cannot be saved this is indicated, and the user prompted for a different name. The database is then closed.

The Database Edit Commands from the menu DBEDCMDS, and the routine called when that command is entered are:

CREATE - calls DBEDCR - Creates a node in the database for the given data name or key.

STORE - calls DBEDST - Obtains a value from the user for the given variable name or key, and puts it in the database at the previously created node. Valid types of data are integer, real, and one-dimensional real array.

DELETE - calls DBEDDL - Removes a variable name(key) from an open database by making the NODE where it is stored unavailable to the user. The NODE is flagged for later compression of the database.

COMMENT - calls DBEDCM - Puts a comment in the database describing the variable. For the present version of DEX where no provision has been made for units, it is desirable to include the

units in the comment. The comment can be up to 64 characters in length.

EXPLAIN - calls DBEDEX - Gets the variable's comment from the database, and displays it at the terminal.

PRINT - calls DBEDPR - Prints the value stored for the given data name.

DUMP - calls DBEDDM - Dumps the current contents of the database to the terminal.

SET-TITL - calls DBEDTS - Prompts the user to enter a database title of up to 64 characters in length, and stores it in the database.

GET-TITL - calls DBEDTG - Gets the title of the database from the database, and displays it at the terminal.

DONE - Returns control to the DEX major control routine DXMAJC which prompts the user to enter a menu item from the menu DEX.MAIN.

A very important feature of the EDIT-DB capability, is that the user can edit a database while in the DEX environment or mode, but outside of the module, so that the user can modify the results generated by a program, before proceeding to the next module used in the design sequence. The DBEDIT routines are a substructure of DEX proper, and they are reached from the DEX mode by entering the menu item EDIT-DB from the menu DEX.MAIN. If the database has not been opened, the user would first enter the menu item OPEN-DB from the menu DEX.MAIN to open the database. The menu item EDIT-DB is one of the DEX commands, and transfers control to the routine DBEDIT, which then prompts the user to enter a menu item from the menu DBEDCMDS, and asks for the name of the datum to be manipulated. The DBEDIT commands are listed above.

After the user has completed editing the database, and has returned to the DEX prompt "enter an item from menu DEX.MAIN" there are many paths that might be taken. If the user has finished using the database, then he would enter the menu item CLOS-DB from the menu DEX.MAIN. DEX will then announce the name of the open database, and ask if the user wants to save it. If the

answer is yes, the user will be given the option of changing the name. The database will then be written to the disk, and the file closed.

4.2 The DEX Database Organization

For the reader to understand the operation of the DEX database, and the further developments made by this investigator, it is necessary to present the formal description of the database structure. The overall view of the database is called its "schema". Martin discussed different levels of description of data: [Martin, 1977]

1. The Global Logical Database Description, or SCHEMA
 - This is the overall view of the logical layout of the data. This is the way that the data is normally perceived.
2. Physical Database Description - A description of the physical layout of the data, and how it is actually stored.

The DEX Database SCHEMA is represented in figure 4.1. This is the overall logical view of the data that would be seen at the terminal if the user issued the DBEDIT command dump, which would dump the contents of the database to the terminal. The physical representation of the data is shown in figures 4.2 - 4.7, and is described in the following sections.

4.2.1 The Physical Description of the DEX Database

The physical description of the DEX Database, or how the data is actually stored, will now be considered. The present version of DEX allows up to 200 variables to be stored in the database. These variables can be one of three types: integer, real, or real array. If the type is real array, then the number of elements in the array and the actual array values are stored in a separate ECS array storage area which is dynamically allocated, and initialized to 2000 words of memory. As the area fills it is expanded in increments of 2000 words. A pointer to the array's location is stored in the value field of the variable in the database.

\$TITLE: **SAMPLE DATABASE

\$	NAME	TYPE	VALUE	COMMENT
	NCREW	(I)	100	NUMBER OF CREW
	DIAMETER	(R)	**UNDEFINED**	HULL DIAMETER (SQUARE FEET)
	POWER	(A)	ARRAY(7)	
	SPEEDAR	(A)	ARRAY(1)	
	LENGTH	(R)	0.46000E+03	

THE ARRAYS

```
$NAME: SPEEDAR , LENGTH= 3 ( 1)
$ 0.50000E+01 0.10000E+02 0.20000E+02
$NAME: POWER , LENGTH= 3 ( 7)
$ 0.80000E+05 0.10000E+06 0.15000E+06
```

Figure 4.1 The Database SCHEMA, and Sample Data

Each individual array can contain up to 200 elements. However, once the array has been stored its length becomes fixed at the number stored, even if it is less than 200.

Figure 4.2 is a representation of the physical layout of the DEX Database.

The actual physical structure of the database is a series of arrays representing the fields of the schema. If the database is an already existing one then the values of the arrays represented in figure 4.2 are loaded from a disk file. If the database is new, then the values of the arrays are initialized in the routine DBINIT when the database is opened. The initial values will be explained below.

The KEY to the database (KEY is the field used to locate entries) is the variable name. The name is up to eight characters in length, and is stored in a two-dimensional array IDENT(I,J). The first four characters of the name are stored in IDENT(I,1), and the second four in IDENT(I,2). The value of the index I indicates the number of the node in the database. The IDENT array is initialized with blanks. The variable type is stored in the array NODTYP(I), which is initialized to all zeros. Next is the array DATUM(I), which will contain the value of the data element. If the variable is an array then the DATUM will contain a pointer to the location of the array in memory. The DATUM array is ini-

TITLE

--

NAVAIL DELCNT MAXECS NEXECS PRECS

1	0	2000	1	0
---	---	------	---	---

BUCKET	1	2	3	4	5	31	32
	0	0	0	0	0	0	0

NODE	IDENT1	IDENT2	NODTYP	DATUM	LINK	LINKF	CMPTR
1			0	0	2	0	0
2			0	0	3	0	0
3			0	0	4	0	0
4			0	0	5	0	0
5			0	0	6	0	0
6			0	0	7	0	0
.
.
.
.
.
199	.	.	0	0	200	0	0
200			0	0	0	0	0

ARRAY STORAGE AREA

--

Figure 4.2 The physical Database Structure

tialized to zero. Next is the array LINK(I), it is initialized to the node number plus one. This is used initially to indicate the next available node in the database. It is later used to link together variables with the same hash value. This will be made clear in an example which will follow. Next is the array CMTPTR(I), which will contain a pointer to a 64 character comment which is stored in the array buffer. The arrays are stored in a separate area of main storage which is obtained dynamically at execution time. The database main structure then is made up of the arrays IDENT(I,1), IDENT(I,2), NODTYP(I), DATUM(I), LINK(I), and CMTPTR(I). The index I is dimensioned 200 in the present version of DEX. In addition to this main 200 node database body there is the auxiliary storage area where the comments and arrays are stored. Also there is a title array which allows for a 64 character title of the database to be stored. The addressing scheme uses a hashing function which converts the name of the datum into a number. The resultant hashed numbers are uniformly distributed between 1 and 32. For example, if the name of the datum was length, then the hashing function would yield the result;

HASH(LENGTH) = 32

The hashing function will be described in more detail in the next section. The result of the hash is used as a pointer to an element in a 32 element array BUCKET(I). The BUCKET is used to contain a pointer to the node in the database where LENGTH is stored. Length is stored in the next available node in the database which is indicated by the variable NAVAIL. NAVAIL is initially set to one. It is updated from the value of the LINK at the node which is currently the next available when the next datum is stored.

4.2.2 The DEX Hashing Function

The DEX hashing function uses the internal machine code values for each character in the KEY name, summing these codes up until the first blank is encountered. The resulting number is truncated using the modulo function to obtain a number between 1 and 32.

4.2.3 An Example of Editing the DEX Database

In this section, an example demonstrating the inserting of several nodes in the database will be given. An example

DEX database editing session is given in Appendix B, and shows the interaction with DEX which was required to enter the variables described in this example in the database. The sample session is given for information only, and will not be discussed as this example is a description of the physical structure of the database. The hashing values used in this example are not necessarily those that would actually result for these variables, but were chosen merely for discussion purposes. Figures 4.3, 4.4, and 4.5 will show how the database is changed as each entry is made. Figure 4.6 will show the effect of deleting an entry from the database. Initially the database is as shown in figure 4.2.

The first name to be entered is LENGTH. Its type is real. It is assumed for this example that for each datum entered DEX has already prompted the user for the datum name and type. This name is hashed and the routine DBHASH returns the value 32. The bucket of 32 is checked to see if there has been a previous entry. Since the value of BUCKET(32) is zero this is the first entry. Since NAVAIL is one, the next available NODE is NODE one. So LENGTH is entered in NODE one as follows: (see figure 4.3)

NODE=1

NAVAIL=LINK(1)=2

LINK(1)=BUCKET(32)=0

BUCKET(32)=NODE=1

IDENT(1,1)=LENG

IDENT(1,2)=TH

NODTYP=-2 The 2 indicates real type and the negative indicates that no data has been entered for this node.

LINKF=-32 The LINKF of the first or only entry in a chain is used to contain the negative value of the BUCKET as a BUCKET pointer. A negative number is used to distinguish the BUCKET pointer from a forward LINK to another NODE.

As a result LENGTH has been entered in NODE one of the database with a NODTYP of -2 to indicate real type with no value entered. The next available node has been set from the link of NODE one, and is NAVAIL=2. The LINK has been updated from the BUCKET to point to any other nodes with the same HASH value. In this case the LINK is zero indicating that there are no others. BUCKET(32) is now pointing to NODE one, and LINKF is pointing back to bucket 32.

TITLE

SAMPLE DATABASE

NAVAIL DELCNT MAXECS NEXECS PRECS

2

0

2000

1

0

BUCKET

1

2

3

4

5

.....

31

32

0

0

0

0

0

.....

0

1

NODE

IDENT1

IDENT2

NODTYP

DATUM

LINK

LINKF

CMTPTR

1

LENG

TH

-2

0

0

-32

0

2

0

0

3

0

0

3

0

0

4

0

0

4

0

0

5

0

0

5

0

0

6

0

0

6

0

0

7

0

0

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

199

0

0

200

0

0

200

0

0

0

0

0

ARRAY STORAGE AREA

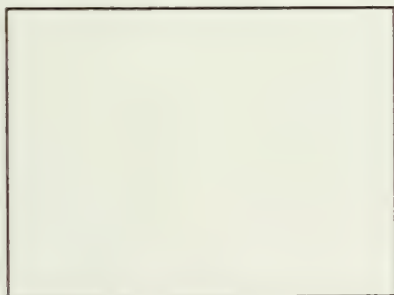


Figure 4.3 Length is entered in the database.

TITLE

SAMPLE DATABASE

NAVAIL DELCNT MAXECS NEXECS PRECS

3	0	2000	1	0
---	---	------	---	---

BUCKET	1	2	3	4	5	...	9	31	32
	0	0	0	0	0	...	2	0	1

NODE	IDENT1	IDENT2	NODTYP	DATUM	LINK	LINKF	CMTPTR
1	LENG	TH	-2	0	0	-32	0
2	DIAM	ETER	-2	0	0	-9	0
3			0	0	4	0	0
4			0	0	5	0	0
5			0	0	6	0	0
6			0	0	7	0	0
.
.
.
.
.
199			0	0	200	0	0
200			0	0	0	0	0

ARRAY STORAGE AREA

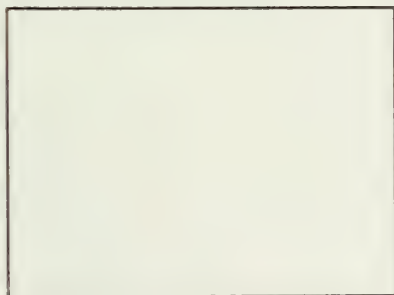


Figure 4.4 Diameter is entered in the database

TITLE

SAMPLE DATABASE

NAVAIL DELCNT MAXECS NEXECS PRECS

5	0	2000	1	0
---	---	------	---	---

BUCKET	1	2	3	4	5	...	9	31	32
	0	4	0	0	0	...	3	0	1

NODE	IDENT1	IDENT2	NODTYP	DATUM	LINK	LINKF	CMTPTR
1	LENG	TH	-2	0	0	-32	0
2	DIAM	ETER	-2	0	0	3	0
3	HEIG	HT	-2	0	2	-9	0
4	SPEE	DAR	-3	5	0	-2	0
5			0	0	6	0	0
6			0	0	7	0	0
.
.
.
.
.
199			0	0	200	0	0
200			0	0	0	0	0

ARRAY STORAGE AREA

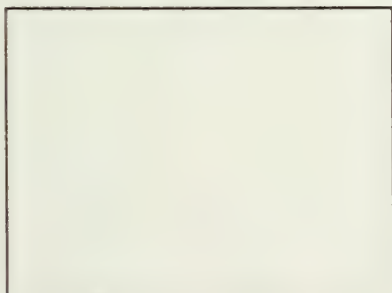


Figure 4.5 Speedar is entered in the database.

TITLE

SAMPLE DATABASE

NAVAIL DELCNT MAXECS NEXECS PRECS

5	1	2000	1	0
---	---	------	---	---

BUCKET	1	2	3	4	5	...	9	31	32
	0	4	0	0	0	...	2	0	1

NODE	IDENT1	IDENT2	NODTYP	DATUM	LINK	LINKF	CMTPTR
1	LENG	TH	-2	0	0	-32	0
2	DIAM	ETER	-2	0	0	-9	0
3	HEIG	HT	-9	0	2	-9	0
4	SPEE	DAR	-3	5	0	-2	0
5			0	0	6	0	0
6			0	0	7	0	0
.
.
.
.
.
199			0	0	200	0	0
200			0	0	0	0	0

ARRAY STORAGE AREA

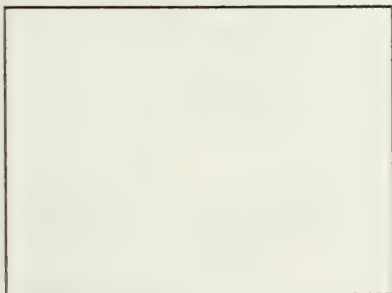


Figure 4.6 Height is deleted from the database.

TITLE

SAMPLE DATABASE

NAVAIL DELCNT MAXECS NEXECS PRECS

7	1	2000	13	1
---	---	------	----	---

BUCKET	1	2	3	4	5	...	9	31	32
	0	4	0	0	6	...	2	5	1

NODE	IDENT1	IDENT2	NODTYP	DATUM	LINK	LINKF	CMTPTR
1	LENG	TH	-2	0	0	-32	0
2	DIAM	ETER	-2	0	0	-9	0
3	HEIG	HT	-9	0	2	-9	0
4	SPEE	DAR	3	1	0	-2	0
5	NCRE	W	1	100	0	-31	0
6	POWE	R	3	7	0	-5	0
.
.
.
.
199			0	0	200	0	0
200			0	0	0	0	0

ARRAY STORAGE AREA

1	3
2	0
3	4
4	0.50000E+01
5	0.10000E+02
6	0.20000E+02
7	3
8	1
9	6
10	0.80000E+05
.	.
.	.

Figure 4.7 Array values are stored.

The next variable to be entered is DIAMETER, which the user has indicated is of real type. Diameter is hashed and the routine DBHASH returns a value of 9. Since NAVAIL=2, DIAMETER will be entered in NODE two as follows:

```
NODE=2
NAVAIL=LINK(2)=3
LINK(2)=BUCKET(9)=0
BUCKET(9)=NODE=2
IDENT(2,1)='DIAM'
IDENT(2,2)='ETER'
NODTYP=-2  Indicating real type.
LINKF(2)=-9
```

As a result DIAMETER has been placed in the database at NODE two. The next available node has been updated from the LINK of NODE two such that the NAVAIL now is three. The LINK(2) has been updated from the BUCKET(9), and is equal to zero. The NODTYP is -2 to indicate a real number will be entered. Since this is the only datum that has hashed to BUCKET(9) so far LINKF(2) is set to point to BUCKET(9).

The next variable entered is HEIGHT which is of real type. HEIGHT is hashed and the routine DBHASH returns a value of 9. When the BUCKET(9) is checked it is found that it contains a pointer to NODE two. This is a collision

since both items cannot occupy NODE two. Since NAVAIL=3, HEIGHT will be entered in NODE three, and will be connected to node two by the pointers LINK and LINKF as follows:

```
NODE=3
NAVAIL=LINK(3)=4
LINK(3)=BUCKET(9)=2
LINKF(2)=NODE=3
BUCKET(9)=NODE=3
IDENT(3,1)='HEIG'
IDENT(3,2)='HT  '
NODTYP=-2  Indicating real type.
LINKF(2)=-9
```

As a result HEIGHT is entered into the database at NODE three, and NAVAIL is set equal to four. This time when LINK(3) is set equal to BUCKET(9), the value 2 is placed in the LINK. A pointer to NODE three is now placed in BUCKET(9). Since both HEIGHT and DIAMETER hashed to BUCKET(9) they are linked together. Thus BUCKET(9) points to HEIGHT in NODE three, NODE three is linked to DIAMETER in NODE two by LINK(3)=2, and DIAMETER in NODE two is linked to HEIGHT in NODE three by LINKF(2)=3. HEIGHT is linked to the BUCKET(9), to which both were hashed, by LINKF(3)=-9.

The next variable to be entered is SPEEDAR, which is a real array with 5 elements. SPEEDAR is hashed and the routine DBHASH returns a value of 2. Since NAVAIL=4, SPEEDAR will be entered in NODE four as follows:

NODE=4

NAVAIL=LINK(4)=5

LINK(4)=BUCKET(2)=0

BUCKET(2)=NODE=4

IDENT(4,1)='SPEE'

IDENT(4,2)='DAR '

NODTYP=-3 Indicating real- array type.

LINKF(2)=-2

DATUM(4)=5 Indicating that 5 array elements will later be stored.

As a result SPEEDAR is entered into the database at NODE four, and the NAVAIL is set to five. LINK(4) is updated from BUCKET(2) and becomes zero. BUCKET(2) is set to point to NODE four. The NODTYP is set equal to -3 which indicates that this entry is an array. The DATUM is set to the number of elements which the user expects to store in the array. When values are entered in the array, the actual number of elements that were entered will be stored followed by pointers to the previous stored array, and a pointer back to the

NODE in the database. Then the actual elements of the array will be stored following these.

At this point HEIGHT will be deleted from the database to show how this is done. The entry is deleted by setting its NODTYP=-9. The LINK of the deleted NODE replaces the LINK of the NODE pointed to by the LINKF of the deleted NODE. The LINKF of the deleted NODE replaces the LINKF of the NODE pointed to by the LINK. If the LINK is zero then this is the end of the chain. If the entry is the first entry after BUCKET, then the value of the deleted LINK is placed in BUCKET. The LINKF is handled as before except its value is the pointer back to the BUCKET. DEX knows that it is a pointer to the BUCKET and not to a NODE because it is negative.

In this example HEIGHT is deleted. So HEIGHT is hashed to BUCKET(9), which points to NODE three. The IDENT of NODE three is compared to the variable name, and since there is a match NODE 3 will be deleted. If there had been no match, each NODE in the chain would be searched for HEIGHT until a match occurred. The links are updated as previously described. The DATUM is set equal to zero, and the NODTYP is set to a minus nine to indicate the NODE has been

deleted. If the NODTYP is a +3 indicating that an array is stored, then the DATUM is set equal to -DATUM. Thus if the NODTYP=-9 and the DATUM is less than zero, then the absolute value of DATUM is the pointer to the array storage block which must be compressed. The array storage area compression capability has not been implemented yet, and is an area for future work. A CMLPTR is handled similarly.

NODE=3

Since there is a LINK the LINKF of that NODE must be modified LINKF(2)=LINKF(3)=-9

BUCKET(9)=LINK(3)=2

DATUM(3)=0 If NODTYP=3 then DATUM(I)=-DATUM(I) so that the pointer to the array storage area is saved for later compression of the array storage area.

CMLPTR(3)=0 If CMLPTR is not 0, CMLPTR(I)=-CMLPTR(I)

LINKF(3)=0

NODTYP=-9 Indicating that this datum has been deleted.

It should be noted by the reader that at this point the variable names LENGTH, DIAMETER, SPEEDAR, and HEIGHT were created as NODEs in the database only, and then HEIGHT was deleted as a NODE in the database. That is to say that the NODEs are created, but no value has been stored for those datum names. To store a value, DEX prompts the user for the name, type, and value. The name is hashed to a BUCKET, and the NODE determined. If the NODE does not exist the user is

so informed. The NODE is checked to see if its IDENT matches the name, if it does then the value is stored in DATUM(NODE). If there is no match the next NODE in the chain is checked until a NODE is found with that name. Once the NODE containing the desired name has been located, its NODTYP is compared to the type indicated by the user. If the type does not match, then the user is informed and the user prompted for a new menu item.

Figure 4.7 shows the database at a later stage after two additional variables and data for several of the variables have been entered. NODE five contains NCREW with a NODTYP of one indicating an integer value. Since the NODTYP is positive a value has been entered for this NODE. NODE six contains a real array POWER. Values have been entered for the two arrays SPEEDAR and POWER. The DATUM for SPEEDAR points to location one in the array storage area, and the DATUM for POWER points to location seven in the array storage area. Looking at the array storage area, the reader will notice that location one contains a three indicating that three numbers are stored. The next entry is a pointer to the previous stored array. This value is zero in this case because this is the first stored array. The next number is a pointer to the NODE four, linking this array back

to the SPEEDAR entry in the database. This is followed by the three speeds which are the array entries. It should be noted that the user had originally said five numbers would be stored, but since only three were stored the size is now constrained to three, the number stored. The next number in the array storage is at location seven which is the beginning of the next array block. This number indicates that three values are stored in the array. Next is the pointer to the previous array which points to location one. This is followed by the pointer to NODE six to which this array belongs. The three array values follow in the next three locations (only one is shown). Looking at the figure 4.7 it is seen that DELCNT = 1 indicating one item has been deleted from the database. MAXECS = 2000 indicating that only one block of ECS storage has been obtained. NEXECS = 13 indicating that the next available location in the ECS area is at location 13. PRECS = 1 indicating that the previous array was stored beginning at location 1. The storage of comments is handled in a similar manner to the arrays.

4.3 Further Developments in the DEX Database

The DEX Database System as described in section 4.2 had the limitation of fixed size. It also lacked the capability

of compressing the database after delete or after some given number of deletes. The DEX Database System handled deletes by making the node available and setting the NODTYP = -9. This technique worked fine, except that it broke up the consistent upward flow of the LINKs, and if there were a large number of deletes would leave a lot of holes in the database. This would be detrimental to any future development of a dynamically extendible database. Another feature of the database is that it had pointers in one direction only. This made it more difficult to locate previous entries in the chain. The decision was made to modify the DEX Database System to be consistent with the possibility of future development in the flexibility of the Database System. To this end, it was decided to add the following capabilities to the already existing database.

1. Bi-directional Pointers - To provide the capability of easily reconnecting the chain of LINKs after a delete. Also to provide possible future flexibility in error recovery. This was done by adding to the physical database structure a forward link, LINKF(I).

2. Modified Delete Structure - The delete technique was modified so that the node is not made available after delete. The next node and previous node in the chain are connected using the bi-directional link structure, and the deleted node made unavailable by putting a -9 in the NODTYP. This provides a consistent link structure with the LINK always pointing backwards in the database, and the LINKF always pointing forward. This facilitates compressing the database.
3. Database Compression Feature - As the database fills up it will have holes left by the now unavailable deleted entries. Therefore, a routine (DBCMPR) was provided to compress the database when it is full or at the users request. This will also facilitate any future development in a dynamically extendible and contractable database.
4. Dynamic Expansion of the Array Storage Area - Utilizing the dynamic memory management routines GETMN and FREEMN, the auxiliary array storage area was made expandable. As the 2000 word storage area fills up, a larger block of storage is obtained.

The current array area is copied into it and the first area released.

4.3.1 The Pointer Structure

The DEX pointer structure has been modified to include the original backward LINK(I), with the addition of the LINKF(I). When collisions occur in the database, and several nodes with the same hash value are chained together, then the direction of the links is as follows. The node address of the last item entered is placed in the bucket, and its LINK points back to the previous node that was entered with that hash value. The LINKF points forward from the previous node to the current one. The links are propagated along the chain in the same manner. The LINKF of the last node entered contains the negative of the bucket address to which it hashed.

4.3.2 The Modified Delete Structure

The delete structure was modified so that it would preserve the clean structure of the links. When a node is

deleted it is not made available but is flagged with a -9 in NODTYP. This will be used by the DBCMPR (compression routine) routine. The next node in a chain and the previous node in the chain have their links joined together by using the LINK and LINKF of the deleted entry to tie them together.

4.3.3 The Database Compression Feature (DBCMPR)

The further capability to compress the database was developed so that as the database fills up, the database can be periodically compressed to open up any holes left by delete. The compress technique used was to search the database until a type of -9 was found. Then a count was updated to keep track of the number of deletes. The nodes were then moved up a number of positions equal to the count until the next -9 was encountered. As the Node is moved up its links were modified including any link to the bucket.

4.3.4 Dynamic Expansion of the Array Storage Area (DBXECS)

With the further development of the DEX dynamic storage allocation achieved by this investigation, it has become possible to begin the development of the routines to allow the database storage to grow with the data. This investigator developed a simple first step routine to expand the database array area as it becomes full. The routine was called DBXECS for Database Expand ECS area. The routine has the following calling sequence,

```
TRUVAL = DBXECS(IDUM)
```

The routine is a logical function with a dummy argument. The required variables are passed in the common DBDAIO which was modified to include the following variables:

MAXECS - is the maximum ECS storage size. It is initialized to 2000 words, and is incremented by 2000 each time the ECS area is expanded.

NEXECS - is a pointer to the next free location in the ECS storage area. (as previously used)

PRECS - is a pointer back to where the previous array is stored in the ECS area.

ECSPTR(2) - is an array containing the address of the old ECS area and the new ECS area.

ECSLEN(2) - is the array containing the length of the two ECS areas. (ECSLEN(1) = MAXECS - 2000, ECSLEN(2) = MAXECS)

The DBXECS routine is called as the array area is filled up. It acquires a new block of main storage whose size is now MAXECS+2000. It saves the current MAXECS and the previous MAXECS. It then copies from ECS area one to ECS area two, frees area one, and then saves the new pointer and maximums. It then returns to the calling routine.

The DBSAVE and DBLOAD routines have been modified so that the final MAXECS, the NEXECS, and the PRECS are saved in the database.

4.4 Future Developments in the DEX Database

The DEX Database System has been developed with the philosophy of the DEX System in mind. That is to provide the user the maximum flexibility while still maintaining consistency of structure, and uniformity of dialogue. The DEX Database System has been designed to use the usual DEX control structure and menu communication system. It has also been designed with its future growth in mind. There are two major areas that need to be considered by future investigators.

The first is to take the database the next step beyond the work of this investigator, and provide a dynamically extendible and contractable database that responds to the size of the database as it grows or items are deleted. This writer has taken a first step in this direction by providing the memory allocation routines, the capability to compress the database after delete (DBCMPR), the capability to expand the array storage area (DBX ECS), bi-directional LINKs, and the enhanced delete capabilities described in the previous section. However, the current version still has a limit of 200 nodes in the main database structure. This limitation needs to be addressed, and a routine developed to make the

database completely expandible. Fagin in his work on "expandible hashing" for dynamic database management has suggested a technique[[Fagin, 1977] that provides a dynamic storage structure that grows and shrinks gracefully as the database grows and shrinks".

The technique chosen by this investigator was to expand the database by acquiring more memory. This does not depend on the use of direct access file management, but has a limitation imposed by the amount of memory required. In the current version of DEX the size of the array storage area is limited by the number of nodes in the database and the size of array, overhead, and comment for each node. This is not an insignificant amount of storage, but is less than the theoretically available sixteen million. If the number of NODES in the database is also made expandible in memory, then the required storage could become unmanageable. Another technique which was considered, and temporarily set aside because of its machine dependencies, is Virtual Storage Access Method (VSAM). This among other things allows direct access files on disk storage. With this procedure only the segments needed at any given time are in memory. Thus storage requirements would be kept at a manageable level. This would facilitate the use of Fagin's technique of hashing to

a segment of the memory and then to a node within the page. The database could then be made totally expandible and contractable without large storage requirements. Both techniques will be experimented with and perhaps a combination of the two will be the best approach. The second area is to provide an alerting capability which allows for a structure that chains dependent variables to other variables upon which it depends. Then alert the user when a change has occurred in an independent variable upon which this variable depends.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

The work of this investigation has provided a further enhancement of the already existing DEX capabilities, and has developed the structure necessary to achieve a truly dynamic database that can respond to the requirements of the complex design problem.

This work has made improvements in the area of dynamic file management which provide a fairly adequate file management structure. Additionally the area of dynamic loading and unloading of modules has been solved by this investigators development of the LOAD, START, and UNLOAD routines utilizing supervisor services macros. The dynamic storage allocation routines and database enhancements have taken the database capabilities a step closer to the concept of an extendible database system that can dynamically respond to the growth of the database.

There are several areas of DEX which still need development. First is the further development of the DEX database system to expand upon the work of this investigator to achieve a dynamically expandible database. Along this line,

the limit of 200 elements in the main body of the database must be addressed and the procedure developed to expand it. Future investigations will include experimentation with both dynamic memory allocation techniques to expand the database in memory, and Direct Access File Management techniques to expand the database in direct access files on disk, bringing only those segments needed into memory. It will be determined which of these is the best procedure, or if a combination of the procedures is the best approach to removing the database size limitations. The development of the dynamic storage allocation routines GETMN and FREEMN have provided the basic tools for this development, along with the modifications to the database system to make it upward compatible with the addition of a consistent bi-directional LINK structure. Finally with the database compress algorithm and the capability to expand the ECS storage area this work has taken the next step in the development of a truly dynamic database.

Second is the implementation of the graphics capabilities into the MIT DEX environment. This capability is the logical next step in the development of DEX as a design system. The decision has been made to pursue segmented graphics as the base for the MIT implementation of DEX graphics. This

will allow the use of segmented graphics routines already implemented in DEX at other installations.

In conclusion DEX then has been developed as a truly Design Executive with a strict adherence to structured programming in its development. DEX follows the structured programming rule of one function for one routine with a consistent top down control structure. DEX allows the user to exercise as much or as little control over the design process as desired. With the dynamic file, memory, and module management capabilities coupled with the removal of restrictions from the DEX Database system and the future addition of a powerful graphics design capability will give the designer the maximum consistency and flexibility in the control of the flow of a design process.

REFERENCES

- Celotto, Richard Charles, Lieutenant, U.S. Navy, An Investigation Into the Use of Databases in Computer-Aided Naval Ship Design, Thesis, Massachusetts Institute of Technology, Cambridge Mass., June 1981
- Donovan, James J., System Programming, McGraw Hill Computer Science Series, McGraw Hill Book Co., New York, 1972
- Fagin, Ronald, Jurg Nivergelt, Nicholas Pippenger, H. Raymond Strong, Extendible Hashing - A Fast Access Method For Dynamic Files, IBM Research Report RJ2305, July 1978
- Herzog, Bertram, "A Transportable Fortran Based Executive System for Computer Aided Ship Design Education", Computer Applications in the Automation of Shipyard Operation and Ship Design II, Editors Jacobsen et al., North-Holland Publishing Company, Amsterdam, 1976, pp. 79-87
- Herzog, B., Module Programmers Guide for the Interactive Computing System DEX at the University of Colorado, revised March 1978
- IBM,1: IBM System/370 Principles of Operation, Form GA22-7000
- IBM,2: OS/VS - DOS/VSE - VM/370 Assembler Language, Form GC33-4010
- IBM,3: OS/VS1 Data Management for System Programmers, Form GC26-3837
- IBM,4: OS/VS1 Data Management Macro Instructions, Form GC26-3872
- IBM,5: OS/VS1 Data Management Services Guide, Form GC26-3874
- IBM,6: IBM Virtual Machine/System Product: System Programmers Guide, Form SC19-6203
- IBM,7: OS/VS1 Supervisor Services and Macro Instructions, Form GC24-5103

IBM,8: OS/VS1 Supervisor Logic, Form SY24-5155

IBM,9: System 370 Reference Summary Card, Form GX20-1850

Osborne, Adam, An Introduction to Microcomputers, Vol. I
Basic Concepts, 2nd edition, Osborne/Mcgraw-Hill,
Berkley Calif., 1980

Madnick, Stuart E., John J. Donovan, Operating Systems,
McGraw Hill Computer Science Series, McGraw Hill Book
Co., New York, 1974

Martin, James, Computer Data-Base Organization, Second Edi-
tion, Prentice-Hall, Inc., Englewood Cliffs, New
Jersey, 1977

APPENDIX A

ASSEMBLY LANGUAGE PROGRAMMING

A-1.0 Why Assembly Language

The assembly language is the symbolic language which is most like the actual machine language. Machine language is the actual numerical codes and addresses stored in memory which the machine hardware translates into action. Depending on the Computer, the machine language will be in Binary (base 2), Octal (base 8), or Hexadecimal (base 16). These numerical machine language codes provide control at the hardware level over machine functions, but are highly burdensome and time-consuming. The advantage of assembly language over machine language is that it allows symbolic representation of instructions and addresses while still allowing just as much control over specific instructions and machine functions.

The advantage of assembly language over high level languages such as Fortran, Cobol, Algol, or PL/I is that the assembly language provides: [Madnick 1974, IBM, 2]

1. Full control over all machine functions.
2. The form closest to machine language while still providing symbolic representation.
3. Ability to override system defaults assumed by the higher level language.
4. Ability to obtain system status information not accessible by higher level languages.
5. Ability to closely control your program down to the byte and even bit level.

These advantages are very important in an executive manager program such as DEX, where it is important to know the status of the system, program modules, attached files, memory management, database management, and DEX requests by the user, so that these elements can be integrated and closely controlled. In this Appendix I will provide background on the general knowledge necessary to writing assembly language code on any machine. I will then address the specific machine structure for the IBM/370 on which this version of DEX is written. I will then address the structure of an

assembly language program on the 370 (the general structure can be adapted to other machines), and explain in greater detail an example assembly language program from the utility routines written in assembly language for DEX.

A-2.0 General Approach to Preparing for a New Computer

When preparing to write assembly language programs on a new computer, the programmer must gain an understanding of the machine's structure in order to control it. Madnick and Donovan [Madnick 1974] suggested a general approach to a new computer that could be adopted by a user wishing to understand the structure and become familiar with a new computer. The procedure consists of a series of questions that should be answered before attempting to write assembly language code on a new machine.

1. Memory

What is the memory's basic unit, size, and addressing scheme?

2. Registers

How many registers are there? What are their sizes, functions, and interrelationships?

3. Data

What type of data can be handled by the computer?
Can it handle characters, numbers, logical data?
How is this data stored?

4. Instructions

What are the classes of instructions on the machine?
Are there arithmetic, logical, symbolic instructions? What are their formats? How are they stored in memory?

5. Special Hardware Features (pertinent to operating systems)

The answers to questions 1,2,3, and 4 will provide sufficient understanding of the machine structure for most users. However, the user who must under-

stand the operating system in order to interface with it, such as the DEX system programmer (not necessary for the DEX module programmer), must understand additional hardware features.

- a. Status of Program - How does the CPU keep track of which instruction to execute? Does the CPU differentiate between operating system state and user state?
- b. Input/Output Processing - How is input and output handled? If separate I/O processors are used, how does the CPU communicate with them?
- c. Interrupts - What kinds of interrupts exist? How are interrupts handled, and can they be intercepted? This is important in DEX for trapping abnormal termination, control information, and other status.
- d. Masking - Can interrupts be ignored or suspended?
- e. Protection - Is there hardware available that can prevent the reading, writing, or execution of parts

of memory? Is it possible to access system routines for DEX use?

f. Timers - Is there a clock on the system that can be accessed?

g. States - Does the CPU have different states? Does the CPU have privileged instructions that may be executed only in certain states?

To Madnick and Donovan's list I would add that the DEX system programmer must also gain an understanding of the operating system control block structure [IBM,3]. Where control blocks are areas of storage in which the system keeps information on data blocks, file status, and executable module status. Also I would recommend that the DEX system programmer become familiar with the system supervisor control program and how to communicate with it. The IBM/370 does this with the SVC (supervisor call) instruction [IBM,4].

For the 370 there are several references which the DEX system programmer should read [Madnick 1974, IBM,1,2,3,4,5].

A-3.0 IBM/370 Specific Machine Structure

In this section I will answer the questions of section A-2.0 as they apply to the IBM/370.

A-3.1 Memory

The IBM/370 is a 32 bit word machine. The basic unit of memory is a byte = 8 bits. A word is composed of 4 bytes and each byte is individually addressable. Therefore each addressable position in memory contains 8 bits of information, where a bit contains either a binary one or zero. Instructions may operate on the following groupings of bytes:

Memory Unit's Name	No. of Bytes	Length in Bits
Byte	1	8
Halfword	2	16
Word	4	32
Doubleword	8	64

The IBM/370 has a total of 2^{24} bytes of memory (about 16 million). The size is based on the addressing scheme used. The 370 uses a 24 bit address for referencing memory, and uses a base/offset addressing scheme. This scheme uses 12 bits of a 32 bit instruction for the offset and 4 more bits to indicate a register that will be used as the base register. Into this base register is placed a 24 bit base or starting address. Another 4 bits of the instruction are used to reference a register used as an index register. The address is then calculated by adding the offset to the contents of the base register. If the storage area is an array the index register can then be used to reference the proper element in the array.

$$\text{Address} = C(\text{base register}) + \text{Offset} + C(\text{index register})$$

where $C(\text{register})$ indicates the contents of register

Any of the 4 bytes in a memory word can be individually addressed, but normally the low order byte is addressed. If the four bytes of the memory word had addresses 94,95,96, and 97 respectively, the memory word would be addressed as 94.

A-3.2 Registers

The 370 registers are of four general types:

- 16 General Purpose 32 bit registers
- 4 Floating Point 64 bit registers
- 16 Special Control 32 bit registers
- 1 Program Status Word 64 bit register

The general purpose registers can be used as base address registers or for arithmetic or logical operations. When used as a base register they contain a starting address, and care should be used not to alter its contents after the desired address is stored. In the case of arithmetic and logical operations these registers act as scratch pads in which calculation results are accumulated, manipulated or compared. The floating point registers are used in floating point calculations where greater accuracy is required. The Program Status Word (PSW) contains the address of the instruction to be executed, as well as protection information, and interrupt status.

The 16 control registers are assigned to special operating system facilities, such as paging or virtual memory handling. They are used, in a fashion similar to the general purpose registers, to act as scratch areas for the system facilities. They are used to hold information specifying if an operation can take place or to furnish and manipulate system information, such as the memory page size, for the control programs. The control registers are referenced by the numbers 0-15 in control instructions such as LOAD CONTROL 13, LOCATION.

There is no interrelationship on the 370 between the 4 types of registers. Within a given register group, such as the general purpose registers, the registers are interrelated by the instructions operating on them.

A-3.3 Data

All data and instructions are stored as binary ones and zeros. However, for convenience these numbers are written in the hexadecimal (base 16) number system. A hexadecimal digit is represented by four binary bits. The relation

between binary, hexadecimal and decimal numbers are given below:

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

So for example the binary number B'0000 0010 1111 1100' has the hexadecimal equivalent X' 0 2 F C '. The bits in a memory word are numbered from left to right, with the left most bit numbered zero and the right most bit numbered thirty one. The IBM/370 allows seven different data formats:

1. Short Form Fixed Point - This format is a 16 bit halfword with bit 0 used as a sign bit. With a

binary zero indicating a positive number, and a binary one indicating a two's complement negative number. A knowledge of binary arithmetic is assumed for this discussion [Osborne,1980]. Bits 1-15 are then an integer number. Some instructions are unsigned and treat the data as a 16 bit unsigned number.

2. Long Form Fixed Point - Similar to the short form except that bits 1-31 contain the integer number.
3. Short Form Floating Point - Is a 32 bit representation, with bit 0 used as a sign bit, bits 1-7 used as an exponent, and bits 8-31 used as a fractional number.
4. Long Form Floating Point - Is a 64 bit representation, with bit 0 used for sign, bits 1-7 for exponent, and bits 8-63 used for the fraction.
5. Decimal-Number Formats - Decimal numbers may be represented in either of two formats, packed decimal or zoned (sometimes referred to as unpacked decimal) decimal. Either of these formats can have

from 1 to 16 bytes, with each byte consisting of two 4 bit codes. The packed decimal format is best suited for decimal arithmetic, while zoned decimal is best suited for input/output.

a. Packed Decimal - In this format each byte contains two 4 bit decimal digits (0-9) with binary representation 0000-1001, except for the right most byte which contains one binary decimal code and one 4 bit sign code in the right most 4 bits. The preferred sign code is 1100 (hex C) for positive, and 1101 (hex D) for negative. There are instructions which perform decimal arithmetic and return the results in the same format. There are also instructions which convert from packed decimal to zoned decimal (unpacked), or from packed decimal to binary.

b. Zoned Decimal - In this format each byte contains a decimal digit in the right 4 bits and a zone code in the left 4 bits. The zone code is a hex F (1111 binary) which can easily be masked out to extract the decimal information.

In this format the zone of the right most byte is treated as a sign code with the same representation used in packed decimal. Zoned decimal digits are part of a larger character set which represents alphanumeric and special characters as two hexadecimal numbers. Thus the zoned decimal numbers are represented by the hex number "F" followed by a decimal digit between 0-9. The zoned decimal numbers are therefore best suited for the input, editing, and output of human-readable numeric data.

6. Logical (Character) Format - In this representation, each character is represented by an 8 bit code [IBM, 9].

A-3.4 Instruction Format

The 370 has five types of instructions and five formats of instructions. The five types are:

1. General Instructions - These are the arithmetic, logical, comparison, branching, memory and register

manipulating, data conversion, and mask and test instructions.

2. Decimal Instructions - These are instructions which manipulate numbers which are in one of the decimal representations.
3. Floating Point Instructions - These are the instructions which normalize, manipulate, and perform floating point arithmetic.
4. Control Instructions - These are the instructions which are used to control and manipulate the supervisor state. These are privileged instructions and cannot be executed from the problem state (that's the user program).
5. Input/Output Instructions - These are the instructions which are used to control the input/output operations on the machine. Generally the program writer will not access these instructions, but will instead access system macros designed to facilitate I/O.

The basic form of all of the different formats of instructions is an 8 bit Op Code, and two operands. In assembly language the 8 bit Op Code is represented with a mnemonic code, such as A for add or L for load. The Op Code operates on the two operands. Generally the flow is from the second operand into the first operand, except in store instructions where the data indicated by the first operand is stored in the location indicated by the second operand. An example of the first type of flow would be:

A 1,2

which would add the contents of register 2 to register 1 and leave the result in register 1. An example of the second type of flow would be:

ST 5,TEMP

which would store the contents of register 5 into the memory location which has the label TEMP. The five formats of instruction are:

1. RR Format (2 bytes in length) - A register-register instruction which has an op code which operates on the contents of two registers.

2. RX Format (4 bytes in length) - A register-indexed memory instruction which moves the contents of a memory location into a register, stores the contents of the register in memory, or performs an arithmetic or logical operation on the contents of the register and the memory location. In this instruction the memory address is found by adding the offset, base, and index.
3. RS Format (4 bytes in length) - A register-memory instruction which is similar to the RX instruction except that it has two register operands and a base/offset memory operand.
4. SI Format (4 bytes in length) - An immediate operand - memory instruction which performs the operation on the memory location and the immediate 8 bit value contained in the instruction.
5. SS Format (6 bytes in length) - A memory-memory instruction which performs the operation on the two locations in memory.

A-3.5 Special Hardware Features

STATUS OF A PROGRAM The program status word (PSW) is the internal register which is the basic control register for the 370. It contains information on interrupt status, what interrupts are masked out of use, whether the machine is in supervisor (privileged) state or problem state (working on a user problem), and the address of the next instruction.

64 bits

System Mask	Key	CMWP	Interrupt code	ILC	CC	Program Mask	Instruction Address
8 bits	4	4	16	2	2	4	24

bit 0

bit 63

The SYSTEM MASK indicates whether or not interrupts will be accepted from a given channel. Bits 0-5 correspond to channels 0-5 and must be on (binary one) if interrupts are to be accepted. Bit 6 set on indicates that interrupts will be accepted from all channels above 6. Bit 7 on indicates that external interrupts are allowed. If the I/O channel is masked (off) then the hardware will hold the interrupt until the interrupt is again enabled.

The KEY is for protection of memory. If an area of memory is referenced its key must match the key in the PSW allowed for the executing program. If the keys do not match a protection exception occurs.

The CMWP field is a collection of 4 flags (C=extended control mode, M=machine check mode, W=wait mode, P=problem state mode). C and M flags are used by the operating system for system control testing, and machine hardware failure checking. They will not be discussed as they are not applicable to the general assembly language programmer. The W bit indicates that the CPU is running (W=0) or waiting (W=1). The P flag indicates the state of the machine (P=0 implies the supervisory state, P=1 implies the user problem is running). The INTERRUPT CODE tells the CPU the type of interrupt last received and thus where to go in memory to handle it. The ILC field is the Instruction Length Code and tells you the length of the previous instruction executed. The CC field is the Condition Code and contains the current value of the condition set by certain instructions. The CC field is tested by the conditional branch instructions, and is set to flag the results of arithmetic and comparison instructions. It might be set to indicate that the result of an operation was positive, negative, or zero. This con-

dition code can then be checked by a conditional branch instruction such as branch on zero (BZ).

The PROGRAM MASK allows the masking of other non I/O interrupts such as underflow and overflow in arithmetic calculations. The INSTRUCTION ADDRESS contains the address of the next instruction to be executed. This field, the condition code, and the previous instruction length code ILC are very useful for debugging programs. The load address of the program can be subtracted (in hex arithmetic) from the address in the PSW at the time of an error to get a relative address of the error.

Of the special hardware features mentioned, the status of program, masking, and protection were covered in the above discussion of the program status word. Input/Output processing and Interrupts will now be discussed.

Input/Output processing involves the transfer of information between the main memory of the computer and an I/O device such as a disk drive. On the IBM 370, I/O devices are attached to I/O processors called CHANNELS, which control all data transfer. The CPU communicates with the I/O channels through the I/O instructions:

Test Channel
Clear Channel
Store Channel ID
Clear I/O
Halt Device
Halt I/O
Start I/O
Start I/O Fast Relief
Test I/O

The first 3 instructions reference a channel only. The last 6 instructions address a channel and a device on that channel. The I/O address is a 16 bit address consisting of two parts. The leftmost eight bits is the channel address, and the rightmost 8 bits is the device address. There are up to 256 channels numbered from 0-255, and their type and assignments are system dependent. Each channel has the facility for addressing up to 256 devices. The CPU uses the START I/O command to identify a channel and tell it to start operation. The channel then loads the Channel Address Word (CAW) from a hardware fixed location in memory. The CAW contains the address in memory of the I/O program the channel is to execute.

The programmer could write an assembly language program to control individual channels and devices. However, the system provides Data Management Macro Instructions which perform I/O operations, and provide the facilities to open, close, and manipulate files [IBM,4]. The reader is referred to the literature for a more detailed description of these facilities. [Madnick, 1974, Donovan, 1982, IBM, 1,2,3,4,5]

Interrupts are hardware signals indicating that some condition requires the immediate attention of the CPU. On the 370 there are five conditions or interrupt types:

Operating Exception - tried to execute an illegal instruction.

Machine Checking - Hardware Failure Detection

External - such as an alarm

Input/Output - channel is finished or waiting to transfer to CPU

Supervisor Call (SVC) - request to the operating system.

An interrupt signal causes a hardware automatic response which notes where execution is when the interrupt occurs, and starts an interrupt action. When the interrupt occurs, the hardware first saves the current PSW in a predefined location in memory. Then the hardware loads the PSW from a specific reserved memory location, which depends on the type of interrupt. Then the CPU executes the interrupt handler program. When the interrupt handler completes its function it restores the old PSW from the fixed location. The CPU resumes execution of the interrupted program at the place where it was interrupted.

A-4.0 Assembly Language Structure

An assembly language program is made up of statements that are either instructions or comments. The assembly language instructions can be divided into three types:

1. Machine Instructions (machine-op)

Which are the symbolic representation of the machine language instruction. An example of a machine instruction would be the add instruction, in which

the CPU is instructed to add the contents of two registers. For example the machine instruction

ADD 1,2

would add the contents of register 1 to the contents of register 2. This type of instruction will be explained more fully in the example program.

2. Assembler Instructions (psuedo-op)

An assembler instruction or psuedo-op is a note to the assembler which is not executable at execution time, but rather tells the assembler how to set up the program while it is assembling it. Examples of this type of instruction would be DC, and DS which define constants or storage. Assembler instructions can also be used to define entry points and other references.

3. Macro Instructions

A macro instruction allows the assembler to substitute a predefined block of code for that macro sym-

bol when it is encountered in the program. This allows the user to define a block of code which is used often in a program as a macro. The name of the macro is then used in the program and the assembler expands it at assembly time. Macros can take arguments as well, and also allows for conditional expansion based on the arguments. The system provides several utility macro definitions for Input/Output handling, data management, and communication with supervisor operations.

There are three types of comments in an assembly language program. In a macro definition comments must begin with a period in column one followed by an *. In the main part of the assembly language program comments on a line by themselves must begin with an * in column one and must not go beyond column 71. Comments may be included on the same line as instructions, but must be after the last field of the instruction and must be separated from the last instruction field by at least one blank.

A-4.1 General Structure of Program

The general structure of an assembly language program depends on the conventions of the particular machine you are using. The basic ideas will be the same as for the IBM/370 but the reader will need to investigate these conventions at his computer facility. Generally psuedo-ops will be used to set up the entry and external linkage to the routine, as well as the starting reference and base register convention. Additionally psuedo-ops will be used to define storage areas. The general convention is that it is the responsibility of the assembly language routine to save all registers at entry, and restore them before returning control to the calling routine. After this setup is accomplished the routine performs its calculations, stores its results, and returns to the calling routine. This structure will be explained in the example, and is summarized below:

- Define program START, ENTRY points, control sectioning (CSECT,DSECT), and external references (EXTRN)

- Identify the base register or registers the program will be USING.
- Save the general purpose registers using the store multiple(SM) instruction into the save area indicated by register 13. The store multiple instruction stores the contents of all the registers from the register indicated by the first operand to that indicated by the second operand into the memory area indicated by the third operand.
- Pick up the arguments if any. The convention for passing arguments will be explained in the section on coding conventions.
- Perform the required calculations and operations.
- Save the results in the arguments.
- Restore the registers by using the load multiple instruction (LM). This is necessary because calculations in your routine have changed the contents of the registers, and this could destroy the results obtained by the calling routine. Also if the registers aren't

restored the return path to the calling routine could be lost.

- Return control to the calling program by branching to the address contained in register 14. (BR 14).
- Use the END psuedo-op to end assembly.

A-4.2 Coding Conventions

The statement field in an assembly language program begins in column 1 and ends in column 71. Column 72 is used to indicate a continuation line follows. If a line is continued then the continuation line begins in column 16. The format of an instruction statement consists of four parts 1) the label, 2) the operation code, 3) the operand entry, and 4) an optional remark. The order is important, and must be from 1-4. The entries must be separated by one or more blanks. If used the label must begin in column 1, and the operation code must start at least one column to the right of column 1. Generally for ease of reading the op-code is placed in column 10, and the operands start in column 16

with the optional remark in any convenient place to the right of the operands.

A-4.3 Subroutine Calling and Return Convention

When calling an assembly language subroutine, the convention is that the addresses of the arguments are loaded consecutively into a parameter list (PLIST), with the address of argument one stored in the first word of storage, and the address of argument two stored in the second word and so on. Then the address of the parameter list is loaded into register one. For example assuming the two arguments NAME and RCODE:

LA	3,NAME	Load address of Name into R3
ST	3,PLIST	Store it in Plist
LA	3,RCODE	Load address of Rcode into R3
ST	3,PLIST+4	Store it in the second word of
PLIST		
LA	1,PLIST	Load the Address of PLIST into
R1		

The convention for passing control is to load the address of the routine you want to branch to into register 15, and the address where you want control to return into register 14. You then branch to the address in register 15. This can be accomplished nicely by use of the BALR (branch and link) instruction, which loads the address of the next sequential instruction into the first operand, and branches to the address contained in the second operand register. For example:

```
L      15,CKADDR
BALR   14,15
CL     0,ZERO
```

In this example the address of the CKFILE routine was previously loaded into the location with the symbol CKADDR. The first instruction loads register 15 from the contents of location CKADDR. The BALR instruction loads the address of the compare logical (CL) instruction which follows it into register 14, and then branches to the address in register 15. Now control has been passed to the CKFILE routine. When the subroutine has completed it's operation it does an unconditional branch to the address contained in register 14 (BR 14) which returns control back to the CL instruction.

This CL instruction then compares the return code in register 0 to zero.

Another convention often used is that register 13 contains the address of a save area for saving register contents.

As the writer of an assembly language subroutine you then can expect the following register conventions when your subroutine receives control.

- Register 15 will contain the address at which your routine has been loaded (since it was used to contain the address of the subroutine in the BALR 14,15 instruction), and thus can be used as the base or reference address register.
- Register one will contain the address of a parameter list where the entries in the parameter list are pointers to the arguments. There are two techniques for obtaining this data. Assuming that there are three arguments, the first technique would be to load multiple into registers 2,3,and 4 from the location pointed to by register 1.

LM 2,4,0(1)

Would result in registers 2,3, and 4 being loaded consecutively from the address pointed to by register 1, C(1)+4, and C(1)+8. Each of these three registers would then be pointing at one of the arguments. To get the argument it is then necessary to load from the address pointed to by the register. For example:

L 5,0(2)

This would load register 5 from the address contained in register 2 plus an offset of zero. The address contained in register two is the address of the first argument, which was loaded into register two by the load multiple instruction above. The other technique is to individually load each register with the address of an argument and then use that register to load another register with the actual argument. For example,

L 2,0(1) load register 2 with the address pointed to by register 1, which is the first address in the parameter list. As a result register 2 will be pointing at the first argument.

L 5,0(2) load register 5 from the location pointed to by register 2. Places the first argument in register 5.

L 3,4(1) load register 3 with the address pointed to by register 1 plus 4. In other words the pointer to the second argument.

L 6,0(3) load register 6 from the location pointed to by register 3.

L 4,8(1) again load register 4 from the location pointed to by register 1 plus 8. Register 4 will contain the address found by adding an offset of 8 bytes to the address of the PLIST which is the address that was loaded into register 1.

L 7,0(4) now register 4 points to the third argument, and this instruction loads it into register 7.

- Register 13 will contain the address of a save area.

- Register 14 will contain the return address in the calling routine.

A-4.4

Macro Organization

Macro definitions are composed of the MACRO psuedo-op indicating that a macro definition follows, a macro name which can take arguments, a body of executable assembly language instructions and conditional assembly macro instructions, and a MEND (Macro END) psuedo-op indicating that the macro definition has ended. Macro definitions must be either at the beginning of a program or in a macro library. Arguments in the macro definition are indicated by & followed by a name. Wherever this occurs in the macro definition the corresponding argument will be substituted for it. Macros can be used for three basic functions.

1. For Text Insertion - This is the most common use of macros. In this fashion macros are used to insert commonly used assembly language instructions into the source program at assembly time. This allows the programmer to not have to continuously write

repetitious code in each program. An example would be a macro to set up the linkage convention between subroutines, such as the DEX IN and OUT macros.

2. For Text Modification - Modify the statements in the macro definition depending on how they are to be used.
3. For Text Manipulation - Utilizing conditional assembly change the way a macro is expanded based on the arguments so that a different block of code is substituted in different situations.

A-4.5 Example Assembly Language Routine

In this section I will consider a specific assembly language subroutine from the DEX assembly language utility routines. The detailed analysis of this routine will hopefully bring the readers understanding of assembly language programming into focus. The routine that I will consider is the dynamic loading routine LOAD. This routine is a logical function. However, the passing of arguments will be the same as for a sub-

routine. The only difference is that in this case the truth value will be returned in register 0, with R0 = 0 indicating .false., and R0 = 1 indicating .true.. The first thing you will encounter when looking at the LOAD routine in figure A-1, is the block of comments explaining its purpose and calling sequence. Comments are important in any programming language to insure that the understanding of the program is passed on, and to facilitate future maintenance and modification. In assembly language comments are absolutely essential in order to obtain any continuity of what was originally done and intended.

After the comment section is the actual program code. The first line of which is the line - LOAD IN CSECT,etc.. This line is a macro definition, and the reader is referred to figure A-2 to see the expansion of the macro. The expanded code is indicated by a + sign in column 1. This is an excellent example of one of the reasons for using macros. All of the DEX assembly language routines have to save the registers and set up the base-using convention. This was done once when the IN macro was written and now is just invoked with each new subroutine. This macro and the OUT macro

which is further down in the code both take arguments which allow for modifying the use of the macro depending on the situation.

Looking at the expanded macro code in figure A-2, it can be seen that the first expanded instruction is the line - LOAD CSECT. In this line is the label LOAD and the psuedo-op CSECT. The CSECT defines to the assembler the beginning of this program as a control section which means it is executable and can be relocated. The routine is given the label LOAD. Another psuedo-op used to indicate a control section is the START psuedo-op, which is used at the start of the first routine in a group of programs to be loaded into memory. All subsequent routines use the CSECT. The reason START was not used is that psuedo-op indicates the first control section in a block of source code. Since this routine is a logical function in DEX it is not the first routine in the source file, the CSECT was used to indicate this is a control section other than the first. A control section is the smallest block of code that can be relocated as a unit.

The next instruction encountered in the macro expansion is a branch to a location which is offset 10 from the contents of register 15. Since register 15 will contain the address of this routine when execution is transferred here the address is 10 (hex A) from the beginning of the program. This is in effect a branch to the STM instruction at statement number 34. This is necessary because the macro expansion uses data constants (DC) at statements 32 and 33, and the CPU can't execute these since they are not instructions. The next instruction is the STM or store multiple which is used to save the registers in the location passed in register 13 + an offset of 12. The order in which the registers are stored in this case is register 14, 15, and then 1-12. After this the LR (load register) instruction is used to copy the contents of register 15 into register 12 so that register 15 can be used for calling other routines. Thus register 12 now becomes the base register, and this is indicated by the USING psuedo-op which is a note to the assembler that it can use register 15 for the base register and it can assume that the address of LOAD will be loaded into it at execution time.

At assembly time the assembler calculates its symbolic addresses relative to the address indicated by the USING psuedo-op. Thus looking at the excerpt from the listing (figure A-3) it can be seen that the address of the symbol NAME at statement 92 is 104 hex relative to LOAD. This then becomes the offset for the symbol NAME. Looking at the reference to NAME in statement number 42, the MVC (move) instruction:

```
MVC    NAME(8),0(2)
```

The instruction says to move 8 bytes of the contents pointed to by register 2 into storage location NAME. Looking to the left of the instruction at the object code column you will find a 12 digit hexadecimal number which is the machine code translation of this instruction:

```
D207 C104 2000
```

The D2 is the op-code, the 07 is the length to be moved less one, the C104 is the first operand address, and the 2000 is the second operand address. Translating the two numbers using the IBM system/370 reference Sum-

mary card [IBM, 6], in the first operand the C is the base register (hex C = decimal 12) and the 104 is the offset. In the second operand the register is number 2 and the offset is 000. Therefore all symbolic references are translated to a base register and the offset from the address in the USING instruction.

Now continuing with the macro expansion of the IN macro, the next group of instructions loads the address of this routine's save area and then chains it to the one passed. The LA instruction at statement 37 loads register 15 with the address of the SAVEAREA argument. The next instruction at statement 38 stores the contents of register 13 into the location determined by adding 4 to the contents of register 15. This loads the address of the save area passed from the routine which called LOAD into LOAD's SAVEAREA+4. The next instruction at statement 39 saves the address of the old save area in SAVEAREA which is pointed to by register 15. Finally the instruction at statement 40 loads register 13 with the address of the new SAVEAREA. The last thing to discuss regarding to IN macro is the argument STORAGE=0. This argument indicates to the IN macro that no dynamic storage is to be allocated. If

desired the programmer could indicate by this argument that a block of main storage is to be allocated for use by this routine at execution of this routine. This completes the group of instructions resulting from the expansion of the macro IN (these are the instructions flagged by the +).

The next instruction at line 41 picks up the first argument in the parameter list by loading register 2 with the pointer in register 1. Then the MVC instruction which was explained in the paragraph on address translation above, moves the contents of the location pointed to by register 2 into the location NAME. The next instruction at line 43 loads the pointer to the third argument (rfcode) into register 3. Then the next MVC instruction moves a literal "1" into the argument. This indicates that if a fatal error occurs it was in the LOAD routine. If no fatal error occurs this code will be reset to zero before returning.

The next group of instructions moves the address of NAME and RCODE into the parameter list and loads the address of the parameter list into register 1. This was discussed previously. Then register 15 is loaded

with the contents of location indicated by the label CKADDR. Looking at statement 100, the label CKADDR is defined using a DC (define constant) psuedo-op, and into this location is placed a V type constant. The V type constant is an external reference which will be resolved at execution time. the V constant tells the assembler to set up an external symbol table with the entry CKFILE. At execution time the address of CKFILE will be known and the address will be placed in this location. Thus the load instruction at line 51 is loading register 15 with the address of the subroutine CKFILE. This and the next two instructions were previously discussed. The BALR 14,15 branches to the subroutine CKFILE because that will be the address loaded into register 15, and then links by placing the return address in register 14. Upon return register 0 will contain a zero if CKFILE is .false. and a one if CKFILE is true. The CL (compare logical) instruction compares the contents of register 0 with the 0 stored in location ZERO. If the comparison is true the CC flag in the PSW is set, and the next instruction BE BOMB is a branch on equal. The BE tests the PSW and if the two were equal it branches to BOMB. This would be the case if the file which we are attempting to load

does not exist. If execution goes to BOMB, the fatal return code in argument 3 is reset to zero by use of the load and move instructions. Then the OUT macro is invoked with the arguments for a return register of zero and a return code of zero indicating a .false. condition. Thus if the routine calling load was an assembly language program, it will test register 0 for the return code. If the calling routine was a fortran routine, then this logical function LOAD will be .false. and the truthflg to which it is equated will be set .false..

If CKFILE was successful in finding the file to be loaded, then the next instruction encountered is the name of a system macro. This macro is called LOAD and takes the argument eploc=NAME. Which passes to the macro the name of the module being loaded. The system LOAD macro is expanded to three instructions. A load address (LA) of the NAME into register 0, a subtract (SR) register 1 from itself to zero it out (this indicates that we are not using a DCB or data control block), and SVC 8. The SVC instruction is a call to the supervisor program in the Operating System which indicates by the code 8 that the module name contained

in register 0 is to be loaded into memory. Upon return from the supervisor call the register 0 will contain the entry point location of the module in memory (the address). The contents of register 0 are then stored in the location ENTRY, and the rfcodes is reset to zero by the same load argument address and move sequence. Also the contents of ENTRY is moved into argument 2. Then the OUT macro is invoked with a return code equal to one indicating that the LOAD was .true.. The OUT macro expansion allows for restoring the registers to the state they were upon entry, and then a BR 14 returns to the calling routine. Where BR 14 is branch to the location in register 14, which is the next instruction in the calling routine. Also in the OUT macro is a LTORG psuedo-op which defines the place to put literal constants. A literal constant is of the form =F'1'. A literal can be used in an instruction as one of the operands, without having to define a storage location yourself. Literals are placed either at the end of the program, or at the location defined by the LTORG instruction.

The last section of the program is the data constant area. In this area there are two types of psuedo-ops

used to reserve storage. The first is the define storage (DS) which reserves the indicated length of storage but does not set it to any initial value. An example of this is the SAVEAREA DS 18F, which saves 18 fullwords of storage with the first labeled SAVEAREA. The other type is the define constant (DC), which sets the storage to the value indicated initially. Valid types are F for fullword H for halfword, X for hexadecimal, V for external address value, C for character, B for binary, and A for address.


```

L      1,TEMP      RESTORE R1
L      2,4(1)      USE R2 FOR THE ADDR OF ARG2
L      3,8(1)      USE R3 FOR THE ADDR OF ARG3
MVC    0(4,2),ENTRY STORE THE ENTRY ADDR IN ARG2
MVC    0(4,3),ZERO  RESET RFCODE = 0, NORMAL TERMINATION
***
      OUT  STORAGE=0,SAVE=SAVEAREA,RETC=1,RETREG=0  RETURN 'TRUE'
***
***** TEXT FILE DID NOT EXIST RETURN FALSE
***
BOMB   L      1,TEMP      RESTORE R1
      L      3,8(1)      USE R3 FOR THE ADDR OF ARG3
      MVC    0(4,3),ZERO  RESET RFCODE = 0, NORMAL TERMINATION
      OUT  STORAGE=0,SAVE=SAVEAREA,RETC=0,RETREG=0  RETURN 'FALSE'
*****
*** DATA CONSTANT AREA
SAVEAREA DS    18F      AREA FOR SAVING THE REGISTERS
TEMP     DS     F       TEMPORARY ONE REGISTER SAVE AREA
NAME     DC     CL8' '   FILENAME
          DC     CL8'TEXT' FILETYPE
          DC     CL2' '   FILEMODE
          DS     OF      ALIGN ON FULL WORD BOUNDARY
RCODE    DS     F       RETURN CODE FROM CKFILE STORED HERE
ENTRY    DS     A       STORAGE AREA FOR MODULE ENTRY ADDRESS
ZERO     DC     4X'00'   ZERO FULL WORD FOR COMPARISON
PLIST    DS     2A       PARAMETER LIST
CKADDR   DC     V(CKFILE) ENTRY POINT ADDRESS FOR CKFILE ROUTINE
END

```

figure A-1


```

1 *****
2 *
3 *      LOAD -
4 *
5 *      SUBROUTINE TO LOAD A MODULE
6 *
7 *      CALLING SEQUENCE -
8 *
9 *          TRUTHVALUE = LOAD(FILENAME,ENTRY,RFCODE)
10 *
11 *      WHERE FILENAME IS THE FILENAME OF MODULE TO BE LOADED.
12 *      WHERE ENTRY IS AN INTEGER ENTRY POINT ADDR OF THE LOADED
13 *      MODULE TO BE RETURNED TO THE CALLING PROGRAM
14 *      WHERE RFCODE IS A FATAL RETURN CODE. ABNORMAL TERMINATION
15 *      IS ASSUMED AND RFCODE IS SET EQUAL TO ONE UPON ENTRY
16 *      TO THIS ROUTINE. IF ABNORMAL TERMINATION OCCURS
17 *      RFCODE = 1 INDICATES THE FAILURE OCCURED HERE. IF THE
18 *      TERMINATION IS NORMAL RFCODE IS SET TO ZERO PRIOR TO
19 *      RETURN TO THE CALLING ROUTINE.
20 *
21 *      TRUTHVALUE IS TRUE IF LOAD WAS SUCCESSFUL AND FALSE IF
22 *      LOAD WAS UNSUCCESSFUL
23 *
24 *      USES THE OS/VS1 SUPERVISOR 'LOAD' MACRO. FOR FURTHER INFO
25 *      SEE 'OS/VS1 SUPERVISOR SERVICES AND MACRO INSTRUCTIONS',
26 *      GC24-5103
27 *
28 *****
29 LOAD      IN      CSECT,STORAGE=0,SAVE=SAVEAREA
30+LOAD      CSECT
31+          B      10(0,15)          BRANCH AROUND ID
32+          DC      AL1(4)           LENGTH OF IDENTIFIER
33+          DC      CL4'LOAD'        IDENTIFIER
34+          STM     14,12,12(13)     SAVE REGISTERS
35+          LR      12,15
36+          USING   LOAD,12
37+          LA      15,SAVEAREA      STATIC SAVE AREA
38+          ST      13,4(15)         CHAIN OLD AREA IN
39+          ST      15,8(13)
40+          LR      13,15
41          L      2,0(1)            LOAD THE ADDR OF ARG1 INTO R2
42          MVC     NAME(8),0(2)     GET LOAD MODULE NAME
43          L      3,8(1)            LOAD THE ADDR OF ARG3 INTO R3
44          MVC     0(4,3),=F'1'     SET RFCODE = 1 LOAD CAUSED ANY FATAL ERR
45          ST      1,TEMP           SAVE R1 UNTIL AFTER LOAD
46          LA      3,NAME           LOAD THE ADDRESS OF NAME INTO R3 AND
47          ST      3,PLIST          STORE IT IN THE PARAMETER LIST
48          LA      3,RCODE          LOAD THE ADDRESS OF THE RETURN CODE IN R3
49          ST      3,PLIST+4        AND STORE IT IN THE PARAMETER LIST

```


50	LA	1,PLIST	LOAD R1 WITH PARAMETER LIST ADDRESS
51	L	15,CKADDR	LOAD THE ENTRY ADDRESS FOR CKFILE
52	BALR	14,15	BRANCH TO ENTRY POINT
53	CL	0,ZERO	COMPARE RETURN CODE IN R0 TO ZERO
54	BE	BOMB	IF RETURN CODE = 0 LOAD MODULE NOT FOUND
55	***		
56	LOAD	EPLOC=NAME	LOAD MODULE (EPLOC = ENTRY POINT LOCATION
57+	LA	0,NAME	LOAD PARAMETER INTO REGISTER ZERO
58+	SR	1,1	SHOW NO DCB PRESENT
59+	SVC	8	ISSUE LOAD SVC
60	ST	0,ENTRY	STORE THE ENTRY POINT ADDR IN ENTRY
61	L	1,TEMP	RESTORE R1
62	L	2,4(1)	USE R2 FOR THE ADDR OF ARG2
63	L	3,8(1)	USE R3 FOR THE ADDR OF ARG3
64	MVC	0(4,2),ENTRY	STORE THE ENTRY ADDR IN ARG2
65	MVC	0(4,3),ZERO	RESET RFCODE = 0, NORMAL TERMINATION
66	***		
67	OUT	STORAGE=0,SAVE=SAVEAREA,RETC=1,RETREG=0	RETURN 'TRUE'
68+	L	13,4(13)	RETURN TO ORIGINAL SAVE AREA
69+	LA	0,1	
70+	ST	0,20(13)	
71+	LM	14,12,12(13)	
72+	BR	14	
73+	LTORG		
74		=F'1'	
75	***		
76	***** TEXT FILE DID NOT EXIST RETURN FALSE		
77	***		
78	BOMB	L 1,TEMP	RESTORE R1
79		L 3,8(1)	USE R3 FOR THE ADDR OF ARG3
80		MVC 0(4,3),ZERO	RESET RFCODE = 0, NORMAL TERMINATION
81		OUT STORAGE=0,SAVE=SAVEAREA,RETC=0,RETREG=0	RETURN 'FALSE'
82+		L 13,4(13)	RETURN TO ORIGINAL SAVE AREA
83+		LA 0,0	
84+		ST 0,20(13)	
85+		LM 14,12,12(13)	
86+		BR 14	
87+		LTORG	
88	*****		
89	*** DATA CONSTANT AREA		
90	SAVEAREA	DS 18F	AREA FOR SAVING THE REGISTERS
91	TEMP	DS F	TEMPORARY ONE REGISTER SAVE AREA
92	NAME	DC CL8' '	FILENAME
93		DC CL8'TEXT'	FILETYPE
94		DC CL2' '	FILEMODE
95		DS OF	ALIGN ON FULL WORD BOUNDARY
96	RCODE	DS F	RETURN CODE FROM CKFILE STORED HERE
97	ENTRY	DS A	STORAGE AREA FOR MODULE ENTRY ADDRESS
98	ZERO	DC 4X'00'	ZERO FULL WORD FOR COMPARISON

99	PLIST	DS	2A	PARAMETER LIST
100	CKADDR	DC	V(CKFILE)	ENTRY POINT ADDRESS FOR CKFILE ROUTINE
101		END		

figure A-2

EXCERPT FROM LISTING

EXTERNAL SYMBOL DICTIONARY PAGE 1

-SYMBOL	TYPE	ID	ADDR	LENGTH	LDID
LOAD	SD	0001	000000	000130	CKFILE ER 0002

PAGE 2

- LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
				29	LOAD	IN CSECT, STORAGE=0, SAVE=SAVEAR
000000				30+	LOAD	CSECT
000000	47F0 F00A	0000A		31+	B	10(0,15)
000004	04			32+	DC	AL1(4)
000005	D3D6C1C4			33+	DC	CL4'LOAD'
000009	00					
00000A	90EC D00C	0000C		34+	STM	14,12,12(13)
00000E	18CF			35+	LR	12,15
			00000	36+	USING	LOAD,12
000010	41F0 C0B8	000B8		37+	LA	15,SAVEAREA
000014	50DF 0004	00004		38+	ST	13,4(15)
000018	50FD 0008	00008		39+	ST	15,8(13)
00001C	18DF			40+	LR	13,15
00001E	5821 0000	00000		41	L	2,0(1)
000022	D207 C104	2000 00104	00000	42	MVC	NAME(8),0(2)
000028	5831 0008	00008		43	L	3,8(1)
00002C	D203 3000	C090 00000	00090	44	MVC	0(4,3),=F'1'
000032	5010 C100	00100		45	ST	1,TEMP
000036	4130 C104	00104		46	LA	3,NAME
00003A	5030 C124	00124		47	ST	3,PLIST
00003E	4130 C118	00118		48	LA	3,RCODE
000042	5030 C128	00128		49	ST	3,PLIST+4
000046	4110 C124	00124		50	LA	1,PLIST
00004A	58F0 C12C	0012C		51	L	15,CKADDR
00004E	05EF			52	BALR	14,15
000050	5500 C120	00120		53	CL	0,ZERO
000054	4780 C094	00094		54	BE	BOMB

PAGE 3

- LOC	OBJECT CODE	ADDR1	ADDR2	STMT	SOURCE	STATEMENT
0				55	***	
				56	LOAD	EPLOC=NAME
000058	4100 C104	00104		57+	LA	0,NAME
00005C	1B11			58+	SR	1,1
00005E	0A08			59+	SVC	8
000060	5000 C11C	0011C		60	ST	0,ENTRY
000064	5810 C100	00100		61	L	1,TEMP
000068	5821 0004	00004		62	L	2,4(1)
00006C	5831 0008	00008		63	L	3,8(1)


```

000070 D203 2000 C11C 00000 0011C      64          MVC    0(4,2),ENTRY
000076 D203 3000 C120 00000 00120      65          MVC    0(4,3),ZERO
66 ***
67          OUT    STORAGE=0,SAVE=SAVEAREA,RETC=1,RETREG=0 RETURN 'TRUE'
00007C 58DD 0004          00004          68+          L      13,4(13)
000080 4100 0001          00001          69+          LA      0,1
000084 500D 0014          00014          70+          ST      0,20(13)
000088 98EC D00C          0000C          71+          LM      14,12,12(13)
00008C 07FE          72+          BR      14
000090          73+          LTORG
000090 00000001          74          =F'1'
75 ***
76 *** TEXT FILE DID NOT EXIST
*** RETURN FALSE
77 ***
000094 5810 C100          00100          78 BOMB      L      1,TEMP
000098 5831 0008          00008          79          L      3,8(1)
00009C D203 3000 C120 00000 00120      80          MVC    0(4,3),ZERO
81          OUT    STORAGE=0,SAVE=SAVEAREA,RETC=0,RETREG=0 RETURN 'FALSE'
0000A2 58DD 0004          00004          82+          L      13,4(13)
0000A6 4100 0000          00000          83+          LA      0,0
0000AA 500D 0014          00014          84+          ST      0,20(13)
0000AE 98EC D00C          0000C          85+          LM      14,12,12(13)
0000B2 07FE          86+          BR      14
0000B8          87+          LTORG
88 *****
89 *** DATA CONSTANT AREA
0000B8          90 SAVEAREA DS      18F
000100          91 TEMP      DS      F
000104 4040404040404040          92 NAME      DC      CL8' '
00010C E3C5E7E340404040          93          DC      CL8'TEXT'
000114 4040          94          DC      CL2' '
000118          95          DS      0F
000118          96 RCODE     DS      F
00011C          97 ENTRY      DS      A
000120 00000000          98 ZERO      DC      4X'00'
000124          99 PLIST      DS      2A
00012C 00000000          100 CKADDR     DC      V(CKFILE)
101          END

```

RELOCATION DICTIONARY PAGE 4

-POS.ID	REL.ID	FLAGS	ADDRESS
0 0001	0002	1C	00012C

CROSS-REFERENCE PAGE 5

-SYMBOL	LEN	VALUE	DEFN	REFERENCES
BOMB	00004	00000094	00078	00054
CKADDR	00004	0000012C	00100	00051
ENTRY	00004	0000011C	00097	00060 00064

LOAD	00001	00000000	00030	00036		
NAME	00008	00000104	00092	00042	00046	00057
PLIST	00004	00000124	00099	00047	00049	00050
RCODE	00004	00000118	00096	00048		
SAVEAREA	00004	000000B8	00090	00037		
TEMP	00004	00000100	00091	00045	00061	00078
ZERO	00001	00000120	00098	00053	00065	00080

LITERAL CROSS-REFERENCE				PAGE	6
-SYMBOL	LEN	VALUE	DEFN	REFERENCES	
O=F'1'	00004	00000090	00074	00044	

figure A-3

APPENDIX B
A Sample DEX Database Editing Session

```
start main
EXECUTION BEGINS...
DEX VERSION 82.1(35 98883)
$DEX AT MIT WELCOMES USER CHRYS DX
$ON 82/138
$AT 14.39.27.00
$THIS IS THE MAY 1982 DEX VERSION
$ENTER AN ITEM FROM MENU - DEX.MAIN
open-db
$ENTER THE NAME OF THE DATA-BASE
$FILE NAME
$ENTER UP TO 24 CHARACTERS
example database<
$DO YOU WISH TO USE A PREVIOUSLY CREATED A DATABASE?
$ENTER AN ITEM FROM MENU - DXYES-NO
no
$ENTER AN ITEM FROM MENU - DEX.MAIN
edit-db
$THE OPEN DATABASE IS NEW
FILE:EXAMPLE DATABASE
$*** NO TITLE ***
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
set-titl
$PLEASE ENTER DATABASE TITLE
$ENTER UP TO 64 CHARACTERS
sample database<
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create
$ENTER TYPE
$ENTER AN ITEM FROM MENU - DB-TYPES
real
$ENTER NAME
length
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create real diameter
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create real height
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create array-rl speedar
$error IN READING AN INTEGER VALUE
```



```

$BAD CHARACTER IS FLAGGED. PRIOR ITEMS ACCEPTED.
$THE BAD VALUE AND SUBSEQUENT ITEMS MUST BE REENTERED
$CREATE ARRAY-RL SPEEDAR
$
5          $ENTER UP TO    1 INTEGER NUMBERS
$ENTER NAME
speedar
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create integer ncrew
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
create
$ENTER TYPE
$ENTER AN ITEM FROM MENU - DB-TYPES
array-rl
$ENTER ARRAY SIZE
$ENTER UP TO    1 INTEGER NUMBERS
9
$ENTER NAME
power
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
store
$ENTER TYPE
$ENTER AN ITEM FROM MENU - DB-TYPES
real
$ENTER NAME
length
$ENTER UP TO    1 REAL NUMBERS
460.
*
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
delete
$ENTER NAME
height
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
store
$ENTER TYPE
$ENTER AN ITEM FROM MENU - DB-TYPES
real
$ENTER NAME
ncrew
$WRONG TYPE
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
store integer ncrew

```



```

$ENTER UP TO    1 INTEGER NUMBERS
100
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
store
$ENTER TYPE
$ENTER AN ITEM FROM MENU - DB-TYPES
array-rl
$ENTER ARRAY SIZE
$ENTER UP TO    1 INTEGER NUMBERS
3
$ENTER NAME
speedar
$ENTER UP TO    3 REAL NUMBERS
5. 10.
$ENTER UP TO    1 ADDITIONAL REAL NUMBERS
20.
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
store array-rl 3 power
$ENTER UP TO    3 REAL NUMBERS
80000. 100000. 150000.
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
print diameter
$DIAMETER
$=UNDEFINED
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
print length
$LENGTH
$    0.460000E+03
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
dump
$TITLE: **SAMPLE DATABASE

```

\$	NAME	TYPE	VALUE	COMMENT
	NCREW	(I)	100	
	DIAMETER	(R)	**UNDEFINED**	
	POWER	(A)	ARRAY(7)	
	SPEEDAR	(A)	ARRAY(1)	
	LENGTH	(R)	0.46000E+03	

THE ARRAYS


```
$NAME: SPEEDAR , LENGTH= 3 ( 1)
$ 0.50000E+01 0.10000E+02 0.20000E+02
$NAME: POWER , LENGTH= 3 ( 7)
$ 0.80000E+05 0.10000E+06 0.15000E+06
$ENTER A DATABASE COMMAND
$ENTER AN ITEM FROM MENU - DBEDCMDS
done
$ENTER AN ITEM FROM MENU - DEX.MAIN
close-db
$THE OPEN DATABASE IS
FILE:EXAMPLE DATABASE
$DO YOU WISH TO SAVE DATABASE?
$ENTER AN ITEM FROM MENU - DXYES-NO
no
$ENTER AN ITEM FROM MENU - DEX.MAIN
quit-dex
```


APPENDIX C

DEX ASSEMBLY LANGUAGE ROUTINES

ABNORM	STM	14, 12, 12(13)	SAVE REGISTERS IN AREA PASSED IN R13	ABT00480
	USING	ABNORM, 12	USE R12 FOR LINKAGE PER HND SVC DOC.	ABT00490
	HND SVC	CLR, 13	CLEAR SVC 13 TRAP	ABT00500
	L	15, ADCON	LOAD R15 WITH POINTER TO DXABND	ABT00510
	BALR	14, 15	BRANCH TO DXABND ROUTINE	ABT00520
	LM	14, 12, 12(13)	RESTORE REGISTERS	ABT00530
	BR	14	RETURN TO CONTROL PROGRAM	ABT00540
		*****	*****	ABT00550
		ADDRESS CONSTANT AREA		ABT00560
		***		ABT00570
		***		ABT00580
ADCON	DC	V(DXABND)	EXTERNAL ROUTINE ADDR CONSTANT	ABT00590
	END			

FILE	ST	6, FILENUMW	STORE IN PARM LIST	ALLO0480
	MVC	FILENAMW(24), O(3)	FILENAME TO BE ALLOCATED	ALLO0490
	LA	1, PLISTW	COMMAND POINTER	ALLO0500
***			*****	ALLO0510
***			THE SUPERVISOR CALL FOR SENDING A COMMAND TO THE SYSTEM	ALLO0520
SVC	SVC	202	EXECUTE COMMAND	ALLO0530
	DC	AL4(BOMB)	ERROR RETURN ADDRESS	ALLO0540
***			*****	ALLO0550
***			SET UP THE RETURN	ALLO0560
***			*****	ALLO0570
	OUT	STORAGE=O, RETC=O, RETREG=O		ALLO0580
BOMB	LA	10, 15		ALLO0590
	OUT	STORAGE=O, RETC=O(10), RETREG=O	RETURN WITH CODE	ALLO0600
***			*****	ALLO0610
***			DATA CONSTANT AREA	ALLO0620
PLIST	DS	OD		ALLO0630
	DC	CL8'FILEDEF'	COMMAND	ALLO0640
FILENUM	DC	CL8', '	UNIT NUMBER	ALLO0650
	DC	CL8'DISK'		ALLO0660
FILENAME	DC	3CL8', '	3 PARTS OF FILE NAME	ALLO0670
	DC	CL8'('	INDICATE START OF OPTION LIST	ALLO0680
	DC	CL8'PERM'	SAVE UNTIL FREED	ALLO0690
	DC	8X'FF'	END OF STRING INDICATOR	ALLO0700
*			*****	ALLO0710
*			*****	ALLO0720
*			*****	ALLO0730
	PLISTW	DS		ALLO0740
	DC	CL8'FILEDEF'	COMMAND	ALLO0750
FILENUMW	DC	CL8', '	UNIT NUMBER	ALLO0760
	DC	CL8'DISK'		ALLO0770
FILENAMEW	DC	3CL8', '	3 PARTS OF FILE NAME	ALLO0780
	DC	CL8'('	INDICATE START OF OPTION LIST	ALLO0790
	DC	CL8'DISP'		ALLO0800
	DC	CL8'MOD'		ALLO0810
	DC	CL8'PERM'		ALLO0820
	DC	8X'FF'	SAVE UNTIL FREED	ALLO0830
*			END OF STRING INDICATOR	ALLO0840
*			*****	ALLO0850
*			*****	ALLO0860
	TERMPARM	DC		ALLO0870
	DC	CL8'FILEDEF', CL8', 'CL8'TERM', CL8'('	'CL8'PERM'	ALLO0880
	DC	CL8'RECFM'		ALLO0890
	DC	CL8'F', CL8'BLKSIZE', CL8'140'		ALLO0900
	DC	CL8'LRECL', CL8'140', 8X'FF'		ALLO0910
			END	ALLO0920
				ALLO0930

	CL	15,=F'36'	CHECK FOR DISK NOT ACCESSED	
	BE	RCD4		CKF00480
	B	ERRUT		CKF00490
RCD1	L	4,=F'1'	SET RCODE = 1	CKF00500
	ST	4,RCODE		CKF00510
	B	ERRUT		CKF00520
RCD2	L	4,=F'2'	SET RCODE = 2	CKF00530
	ST	4,RCODE		CKF00540
	B	ERRUT		CKF00550
RCD3	L	4,=F'3'	SET RCODE = 3	CKF00560
	ST	4,RCODE		CKF00570
	B	ERRUT		CKF00580
RCD4	L	4,=F'4'	SET RCODE = 4	CKF00590
	ST	4,RCODE		CKF00600
*****	*****	*****	*****	CKF00610
ERRUT	MVC	O(4,3),RCODE	MOVE RCODE INTO ARG LIST	CKF00620
	OUT	STORAGE=O,SAVE=SAVE,RETC=O,RETREG=O	RETURN 'FALSE'	CKF00630
*****	*****	*****	*****	CKF00640
***	DATA STORAGE AREA			CKF00650
***				CKF00660
SAVE	DS	18F	REGISTER SAVE AREA	CKF00670
RCODE	DS	F	RETURN CODE	CKF00680
TEMP	DS	F	TEMPORARY WORD OF STORAGE	CKF00690
	END			CKF00700
				CKF00710

LA	10,8(2,11)	ADDRESS OF UNIT ENTRY	CL000480
USING	UNITNTRY,10	ESTABLISH ADDRESSABILITY	CL000490
L	9,UABLKAD	ADDRESS OF UNIT BLOCK	CL000500
LA	5,1	GET A ONE FOR PROCESSING USE	CL000510
CR	9,5	IS THE UNIT IN USE?	CL000520
BE	DONE	NO, THEN WE DON'T HAVE TO CLOSE IT	CL000530
LA	8,76(.9)	POINT TO DCB IN UNIT BLOCK	CL000540
CLOSE	((8))	AND CLOSE IT	CL000550
FREEMAIN	R,LV=164,A=(9)	AND RETURN THE STORAGE	CL000560
ST	5,UABLKAD	AND RESET UNIT ENTRY TO SHOW 'UNUSED'	CL000570
DONE	OUT	STORAGE=0,RETC=0,SAVE=SAVEAREA OK, WE'RE SUCCESSFUL	CL000580
NOCANDO	OUT	STORAGE=0,RETC=4,SAVE=SAVEAREA OOPS CAN'T DO IT	CL000590
MODIADD	DC	V(IHNUATBL)	CL000600
MODIADD	DC	V(IHOUATBL)	CL000610
SAVEAREA	DS	72C	CL000620
UATBL	DSECT		CL000630
UAPREFIX	DS	OD	CL000640
UACURRNT	DS	H	CL000650
UATABLEN	DS	H	CL000660
DEFDSRNS	DS	4X	CL000670
UNITNTRY	EQU	*	CL000680
UABLKAD	DS	A	CL000690
DSA	DSECT		CL000700
END			CL000710

		CURRENT LUN IN PROGRESS??	
		NUMBER OF ENTRIES IN TABLE	
		DSRNS FOR DEFAULT UNITS(MSG.READ,PRINT,PUN)	
		ADDRESS OF UNIT BLOCK, OR '1' IF UNUSED	


```

***** TITLE 'JGV MID - OBTAIN THE VM USERID FOR THE PERSON USING DEX' JGV00010
*****
*
* ROUTINE TO RETURN THE VM USERID JGV00020
* CALL JGV MID(V MID) JGV00030
*
* WHERE V MID WILL BE 8 BYTES, EBCDIC JGV00040
* J. GRAHAM / USER SERVICES JGV00050
* DATE: 02/28/78 JGV00060
* JGV00070
* JGV00080
* JGV00090
* JGV MID START O JGV00100
* STM 14,12,12(13) SAVE REGISTERS JGV00110
* BALR 12,0 ESTABLISH BASE JGV00120
* USING *,12 REGISTER JGV00130
*
* L 2,0(1) MOVE 1ST ARG ADDRESS TO R2 JGV00140
* LA 6,RETAREA JGV00150
* LA 10,CMM DL (MAX) NO. OF BYTES TO BE STORED JGV00160
* DC X'83',X'6A',XL2'0000' JGV00170
* MVC O(8,2),USERID MOVE ID INTO CALL LIST JGV00180
*
* LM 2,12,28(13) RESTORE REGISTERS JGV00190
* MVI 12(13),X'FF' INDICATE CONTROL RETURNED TO CALLING PROG JGV00200
* BCR 15,14 RETURN TO CALLING PROGRAM JGV00210
*
* DATA STORAGE AREA JGV00220
*
* DS OD JGV00230
* SAVEAREA DS 18F JGV00240
* RETAREA DS 3D JGV00250
* CMM DL EQU *-RETAREA JGV00260
* BUFFER ORG RETAREA JGV00270
* DUMMY DS XL16 JGV00280
* USERID DS CL8 JGV00290
* END JGV00300
JGV00310
JGV00320
JGV00330
JGV00340

```



```

LOAD EPLOC=NAME                                LOAD MODULE (EPLOC = ENTRY POINT LOCATION)
ST O,ENTRY                                     STORE THE ENTRY POINT ADDR IN ENTRY
L 1,TEMP                                       RESTORE R1
L 2,4(1)                                       USE R2 FOR THE ADDR OF ARG2
L 3,8(1)                                       USE R3 FOR THE ADDR OF ARG3
MVC O(4,2),ENTRY                             STORE THE ENTRY ADDR IN ARG2
MVC O(4,3),ZERO                              RESET RFCODE = 0, NORMAL TERMINATION

***
OUT STORAGE=O,SAVE=SAVEAREA,RETC=1,RETREG=O  RETURN 'TRUE'

***
*****
TEXT FILE DID NOT EXIST RETURN FALSE

***
BOMB
L 1,TEMP                                       RESTORE R1
L 3,8(1)                                       USE R3 FOR THE ADDR OF ARG3
MVC O(4,3),ZERO                              RESET RFCODE = 0, NORMAL TERMINATION
OUT STORAGE=O,SAVE=SAVEAREA,RETC=O,RETREG=O  RETURN 'FALSE'

*****
*** DATA CONSTANT AREA
SAVEAREA DS 18F
TEMP DS F
NAME DC CL8', '
DC CL8'TEXT',
DC CL2', '
DC OF
RCODE DS F
ENTRY DS A
ZERO DC 4X'00'
PLIST DS 2A
CKADDR DC V(CKFILE)

AREA FOR SAVING THE REGISTERS
TEMPORARY ONE REGISTER SAVE AREA
FILENAME
FILETYPE
FILEMODE
ALIGN ON FULL WORD BOUNDARY
RETURN CODE FROM CKFILE STORED HERE
STORAGE AREA FOR MODULE ENTRY ADDRESS
ZERO FULL WORD FOR COMPARISON
PARAMETER LIST
ENTRY POINT ADDRESS FOR CKFILE ROUTINE

END
LOAO0480
LOAO0490
LOAO0500
LOAO0510
LOAO0520
LOAO0530
LOAO0540
LOAO0550
LOAO0560
LOAO0570
LOAO0580
LOAO0590
LOAO0600
LOAO0610
LOAO0620
LOAO0630
LOAO0640
LOAO0650
LOAO0660
LOAO0670
LOAO0680
LOAO0690
LOAO0700
LOAO0710
LOAO0720
LOAO0730
LOAO0740
LOAO0750
LOAO0760
LOAO0770

```



```

L      3,UNIT                                LOAD FTOO INTO R3
ICM    3,B'0011',DECIMAL+6                INSERT THE LAST 2 ZONED DIGITS
ST     3,DCBDDNAM                          MODIFY DCB NAME FOR RDJFCB
L      5,UNIT+4
ST     5,DCBDDNAM+4
*****
*** READ JFCB AND CHECK FOR EXISTANCE OF THE FILEDEF
***
L      11,JFCBAREA
USING IHAJFCB,11
ST     1,TEMP
LA     1,MSGOFF
SVC    202
DC     AL4(++4)
RDJFCB MF=(E,EX1)
CLC    FCBRD(4),ZERO
BE      NOFILE
*****
*** NO ERROR, RETURN QUERY2 = 'TRUE' AND THE FILEDEF INFO
***
DEVTYPE DCBDDNAM,DTYPE
L      1,TEMP
L      3,4(1)
L      4,8(1)
L      5,12(1)
L      6,16(1)
L      7,20(1)
MVC    O(8,3),=F'O'
MVC    O(8,4),FCBDSNAM
MVC    O(8,5),FCBDSTYP
MVC    O(2,6),FCBDSMD
***
L      8,DTYPE
N      8,MASK
ST     8,DTYPE
CLC    DTYPE(4),DEV1
BNE    NEXT1
MVC    O(8,7),PRINTR
B      OUT
***
NEXT1
CLC    DTYPE(4),DEV2
BNE    NEXT2
MVC    O(8,7),READER
B      OUT
***
NEXT2
CLC    DTYPE(4),DEV3
BNE    NEXT3

```

CHECK IF DEVICE CODE IS A PRINTER
 IF NO MATCH CHECK NEXT TYPE
 IF MATCH STORE PRINTER IN ARG6
 AND RETURN

CHECK IF DEVICE CODE IS A READER
 IF NO MATCH CHECK NEXT TYPE
 IF MATCH STORE READER IN ARG6
 AND RETURN

CHECK IF DEVICE CODE IS A TERMINAL
 IF NO MATCH CHECK NEXT TYPE


```

***
NEXT3      MVC      O(8,7),TERMINL
            B        OUT
            CLC      DTYPE(4),DEV4
            BNE      NEXT4
            MVC      O(8,7),TAPE
            B        OUT
            CLC      DTYPE(4),DEV5
            BNE      NEXT5
            MVC      O(8,7),DISK
            B        OUT
            CLC      DTYPE(4),DEV6
            BNE      NEXT6
            MVC      O(8,7),PUNCH
            B        OUT
            CLC      DTYPE(4),DEV7
            BNE      OUT
            MVC      O(8,7),CRT
            B        OUT
            *****
            *** NO FILEDEF, RETURN FLAG = 0 AND BLANK FILE INFORMATION
            *****
            NOFILE   SR      4,4
                     L      1,TEMP
                     L      3,4(1)
                     L      5,8(1)
                     L      6,12(1)
                     L      7,16(1)
                     L      8,20(1)
                     ST     4,0(3)
                     ST     4,0(5)
                     ST     4,4(5)
                     ST     4,0(6)
                     ST     4,4(6)
                     ST     4,0(7)
                     ST     4,0(8)
                     ST     4,4(8)
            *****
            LA      1,MSGON
            SVC      202
            DC      AL4(****)
            OUT     STORAGE=O,SAVE=SAVEAREA,RETC=O,RETREG=O RETURN 'FALSE'
            *****
            OUT     LA      1,MSGON
            *****
            IF MATCH STORE TERMINAL IN ARG6
            AND RETURN
            CHECK IF DEVICE CODE IS A TAPE
            IF NO MATCH CHECK NEXT TYPE
            IF MATCH STORE TAPE IN ARG6
            AND RETURN
            CHECK IF DEVICE CODE IS A DISK
            IF NO MATCH CHECK NEXT TYPE
            IF MATCH STORE DISK IN ARG6
            AND RETURN
            CHECK IF DEVICE CODE IS A PUNCH
            IF NO MATCH CHECK NEXT TYPE
            IF MATCH STORE PUNCH IN ARG6
            AND RETURN
            CHECK IF DEVICE CODE IS A CRT
            IF NO MATCH DONE
            IF MATCH STORE CRT IN ARG6
            AND RETURN
            *****
            UNIT NOT ASSIGNED RETURN ARG2=0
            RESTORE R1 AFTER RDJFCB MACRO
            USE R3 FOR THE ADDR OF ARG2
            USE R5 FOR THE ADDR OF ARG3
            USE R6 FOR THE ADDR OF ARG4
            USE R7 FOR THE ADDR OF ARG5
            USE R8 FOR THE ADDR OF ARG6
            STORE O IN RFCODE -> NOT FATAL
            ZERO OUT FILE NAME
            ZERO OUT FILE TYPE
            ZERO OUT FILE MODE
            ZERO OUT DEVICE
            PREPARE TO ISSUE CP COMMAND TO
            RESTORE EMSG AT TERMINAL TO TEXT
            IF ERROR CONTINUE WITH NEXT INST
            STORAGE=O,SAVE=SAVEAREA,RETC=O,RETREG=O RETURN 'FALSE'
            *****
            PREPARE TO ISSUE CP COMMAND TO
            *****

```


SVC	202		RESTORE MSG AT TERMINAL TO TEXT	QUEO1420
DC	AL4(****)		IF ERROR CONTINUE WITH NEXT INST	QUEO1430
OUT	STORAGE=O,SAVE=SAVEAREA,RETC=1,RETREG=O RETURN 'TRUE'			QUEO1440
*****	*****		*****	QUEO1450
EX1	RDJFCB (FILEDEF.),MF=L			QUEO1460
*****	*****		*****	QUEO1470
***	DATA CONSTANT AREA			QUEO1480
SAVEAREA	DS 18F		SAVE AREA FOR REGISTERS	QUEO1490
DTYPE	DS 2F		RETURN AREA FOR DEVTYPE MACRO	QUEO1500
TEMP	DS F		TEMPORARY STORAGE AREA	QUEO1510
DCODE	DS F		DEVICE CODE	QUEO1520
DECIMAL	DS D		TEMP STORAGE FOR DECIMAL CONVERT	QUEO1530
MASK	DC X'0000FFFF'		MASK FOR EXTRACTING DEVICE CODE	QUEO1540
UNIT	DC C'FT00FOO1'		AREA FOR CONSTRUCTING UNIT NO.	QUEO1550
ZERO	DC X'00000000'		ZERO WORD FOR COMPARISON	QUEO1560
***				QUEO1570
DEVLIST	DS OF		LIST OF DEVICES TO RETURN IN DEV	QUEO1580
PRINTR	DC CL8'PRINTER'			QUEO1590
READER	DC CL8'READER'			QUEO1600
TERMN1	DC CL8'TERMINAL'			QUEO1610
TAPE	DC CL8'TAPE'			QUEO1620
DISK	DC CL8'DISK'			QUEO1630
PUNCH	DC CL8'PUNCH'			QUEO1640
CRT	DC CL8'CRT'			QUEO1650
***				QUEO1660
DEV1	DC X'00000808'		DEVICE CODE FOR A PRINTER	QUEO1670
DEV2	DC X'00000801'		DEVICE CODE FOR A READER	QUEO1680
DEV3	DC X'00000820'		DEVICE CODE FOR A TERMINAL	QUEO1690
DEV4	DC X'000008001'		DEVICE CODE FOR A TAPE	QUEO1700
DEV5	DC X'00002008'		DEVICE CODE FOR A DISK	QUEO1710
DEV6	DC X'00000802'		DEVICE CODE FOR A PUNCH	QUEO1720
DEV7	DC X'00001C'		DEVICE CODE FOR A CRT	QUEO1730
***				QUEO1740
MSGOFF	DS OD		STORAGE FOR CP COMMAND TO TURN	QUEO1750
	DC CL8'CP'		ERROR MESSAGE OFF AT TERMINAL	QUEO1760
	DC CL8'SET'			QUEO1770
	DC CL8'EMSG'			QUEO1780
	DC CL8'OFF'			QUEO1790
	DC 8X'FF'			QUEO1800
***				QUEO1810
MSGON	DS OD		STORAGE FOR CP COMMAND TO RESTORE	QUEO1820
	DC CL8'CP'		ERROR MESSAGE TO TEXT AT TERMINAL	QUEO1830
	DC CL8'SET'			QUEO1840
	DC CL8'EMSG'			QUEO1850
	DC CL8'TEXT'			QUEO1860
	DC 8X'FF'			QUEO1870
*****	*****		*****	QUEO1880


```

*** DATA CONTROL BLOCK
FILEDEF DCB DSORG=DA,EXLST=LST,MACRF=(RI),DDNAME=DDN
*****
*** AREA INTO WHICH RDJFCB MACRO DUMPS SYSTEM DATA CONTROL BLOCK
LST DS OF
DC X'07'
DC AL3(JFCBAREA)
JFCBAREA DS OF,176C
*****
***** CMSCB *****
***** MACRO FOR DSECT TO MAP JFCB *****
***** MACRO FOR DSECT TO MAP DCB *****
***** DCBD DSORG=PS *****
***** END *****

```

```

QUE01890
QUE01900
QUE01910
QUE01920
QUE01930
QUE01940
QUE01950
QUE01960
QUE01970
QUE01980
QUE01990
QUE02000
QUE02010

```



```

***** TITLE 'READEC AND WRITEC ROUTINES TO SIMULATE CDC ECS STORAGE' REAO0010
***** READEC,WRITEC REAO0020
***** SUBROUTINES TO SIMULATE CDC ECS REAO0030
***** REAO0040
***** REAO0050
***** REAO0060
***** REAO0070
***** REAO0080
***** REAO0090
***** REAO0100
***** 'READS' LENWDS WORDS OF 'ECS', STARTING AT OFFSETFROM ECSPTR REAO0110
***** INTO STRING, STARTING AT BYTE ONE REAO0120
***** REAO0130
***** ECSPTR IS THE POINTER TO THE BEGINING OF THE ECS AREA REAO0140
***** REAO0150
***** ECSLEN IS THE LENGTH OF THE ECS AREA REAO0160
***** REAO0170
***** CALL WRITEC(STRING.OFFSET,LENWDS,ECSPTR,ECSLEN) REAO0180
***** REAO0190
***** OPPOSITE OF READEC, SAME ARGS, BUT MOVE IS FROM STRING TO 'ECS' REAO0200
***** REAO0210
***** REAO0220
***** REAO0230
***** REAO0240
***** REAO0250
***** REAO0260
***** REAO0270
***** REAO0280
***** REAO0290
***** REAO0300
***** REAO0310
***** REAO0320
***** REAO0330
***** REAO0340
***** REAO0350
***** REAO0360
***** REAO0370
***** REAO0380
***** REAO0390
***** REAO0400
***** REAO0410
***** REAO0420
***** REAO0430
***** REAO0440
***** REAO0450
***** REAO0460
***** REAO0470

READEC IN ENTRY,STORAGE=0
LM 2,6,0(1) PICK UP ADDRESS OF ARGS

R2 = STRING
R3 = OFFSET
R4 = LENWDS
R5 = ECSPTR
R6 = ECSLEN

MVC ECSPTR(4),0(5) LOAD ECSPTR
MVC ECSLEN(4),0(6) LOAD ECSLEN
L 3,0(3) CGET OFFSET INTO ECS
SLL 3,2 ... IN BYTES
C 3,ECSLEN CHECK IF VALID
BH RNOGOOD
L 5,ECSPTR
L 4,0(4)
SLL 4,2
BCTR 4,0 MINUS ONE FOR HARDWARE
LA 5,0(3,5) ADJUST ECSADD WITH OFFSET
EX 4,RMVC MOVE INTO STRING

*****
* OUT STORAGE=0

```



```

* RMVC MVC O(0,2),O(5) REA00480
* RNOGOOD OUT STORAGE=0,RETC=4 BAD RETURN REA00490
*** REA00500
*** REA00510
*** REA00520
*** REA00530
*** REA00540
*** REA00550
*** REA00560
*** REA00570
*** REA00580
*** REA00590
*** REA00600
*** REA00610
*** REA00620
*** REA00630
*** REA00640
*** REA00650
*** REA00660
*** REA00670
*** REA00680
*** REA00690
*** REA00700
*** REA00710
*** REA00720
*** REA00730
*** REA00740
*** REA00750
*** REA00760
*** REA00770
*** REA00780
*** REA00790
*** REA00800
*** REA00810
*** REA00820
*** REA00830
*** REA00840
*** REA00850
*** REA00860
*** REA00870
*** REA00880
*** REA00890
*** REA00900
*** REA00910
*** REA00920

*****
*** ROUTINE WRITEC ENTRY POINT
***
*** WRITEC IN ENTRY, STORAGE=0
LM 2,6,O(1) PICK UP ADDRESS OF ARGS
*
* R2 = STRING
* R3 = OFFSET
* R4 = LENWDS
* R5 = ECSPTR
* R6 = ECSLEN

MVC ECSPTR(4),O(5) LOAD ECSPTR
MVC ECSLEN(4),O(6) LOAD ECSLEN
L 3,O(3) GET OFFSET
SLL 3,2 ...IN BYTES
C 3,ECSLEN CHECK FOR TOO BIG
BH WNOGOOD
L 5,ECSPTR
L 4,O(4)
SLL 4,2
BCTR 4,0
LA 5,O(3,5)
EX 4,W MVC BASE FOR ECS
GET LENGTH
...IN BYTES
MINUS ONE FOR HARDWARE
ADJUST ECS ADDR FOR OFFSET
STORE STRING INTO ECS

* -----*
*
* OUT STORAGE=0
*
* WNOGOOD OUT STORAGE=0,RETC=4 BAD RETURN
*
* WMVC MVC O(0,5),O(2)
*****
*** DATA CONSTANT AREA
***
*** ECSPTR DS A POINTER TO BEGINNING OF ECS STORAGE AREA
*** ECSLEN DC A(O) LENGTH OF THE ECS STORAGE AREA
*** END

```



```

*****
***** TITLE 'SYSTEM - EXIT TO CMS SUBSET' *****
*****
*
* SYSTEM -
*
* SUBROUTINE TO EXIT TO THE CMS SUBSET (OPERATING SYSTEM)
*
* CALLING SEQUENCE,
*
* CALL SYSTEM
*
* EXITS TO THE CMS OPERATING SYSTEM SUBSET TO ALLOW TRANSIENT
* CMS COMMANDS. TO RETURN TO DEX TYPE RETURN.
*
*****
*
* SYSTEM IN CSECT, STORAGE=0, SAVE=SAVEAREA
* LA 1,PARMS SET UP FOR THE SVC CALL
* CNOP 2,4
* SVC 202 USE THE SVC TO INVOKE THE COMMAND IN PARMS
* DC A(OUT)
*
* OUT OUT STORAGE=0, SAVE=SAVEAREA
*
* *** DATA STORAGE AREA
*
*
* DS OD START STORAGE ON DOUBLEWORD BOUNDARY
* DC CL8'SUBSET' COMMAND FOR CMS SUBSET
* DC 8X'FF' ENDS PARAMETERS
*
* SAVEAREA DS 18F REGISTER SAVE AREA
*
* END
*****
SYSO0010
SYSO0020
SYSO0030
SYSO0040
SYSO0050
SYSO0060
SYSO0070
SYSO0080
SYSO0090
SYSO0100
SYSO0110
SYSO0120
SYSO0130
SYSO0140
SYSO0150
SYSO0160
SYSO0170
SYSO0180
SYSO0190
SYSO0200
SYSO0210
SYSO0220
SYSO0230
SYSO0240
SYSO0250
SYSO0260
SYSO0270
SYSO0280
SYSO0290
SYSO0300

```


TIM00010	*****	TITLE 'TIME - GETS TIME AND DATE FROM SYSTEM'	*****	TIM00010
TIM00020	*			TIM00020
TIM00030	*			TIM00030
TIM00040	*	TIME		TIM00040
TIM00050	*			TIM00050
TIM00060	*	GETS THE TIME AND DATE FROM THE SYSTEM		TIM00060
TIM00070	*			TIM00070
TIM00080	*	CALLING SEQUENCE		TIM00080
TIM00090	*			TIM00090
TIM00100	*	CALL TIME(STRING)		TIM00100
TIM00110	*			TIM00110
TIM00120	*	RETURN TIME AND DATE IN STRING IN THE FORM 'HH.MM.SS.TH YY/DDD'		TIM00120
TIM00130	*			TIM00130
TIM00140	*	USES TIME SVC - FOR MORE INFO, SEE 'SUPERVISOR SERVICES AND		TIM00140
TIM00150	*	MACRO INSTRUCTIONS', IBM GC24-5103		TIM00150
TIM00160	*	*****		TIM00160
TIM00170				TIM00170
TIM00180		TIME IN CSECT, STORAGE=128		TIM00180
TIM00190		L 11,0(1)		TIM00190
TIM00200		USING MSG,11		TIM00200
TIM00210		MVC HH(16),CHARS		TIM00210
TIM00220		TIME		TIM00220
TIM00230		ST 1,DATE		TIM00230
TIM00240		ST O.TIMER		TIM00240
TIM00250		UNPK UNDATE(5),DATE(4)		TIM00250
TIM00260		UNPK UNTIME(9),TIMER(5)		TIM00260
TIM00270		MVC HH(2),UNTIME		TIM00270
TIM00280		MVC MM(2),UNTIME+2		TIM00280
TIM00290		MVC SS(2),UNTIME+4		TIM00290
TIM00300		MVC TH(2),UNTIME+6		TIM00300
TIM00310		MVC YY(2),UNDATE		TIM00310
TIM00320		MVC DDD(3),UNDATE+2		TIM00320
TIM00330		OUT STORAGE=128		TIM00330
TIM00340		CHARS DC C'HH.MM.SS.TH YY/DDD'		TIM00340
TIM00350		DSA DSECT		TIM00350
TIM00360		SAVEAREA DS 72C		TIM00360
TIM00370		DATE DS F		TIM00370
TIM00380		TIMER DS F		TIM00380
TIM00390		UNDATE DS D		TIM00390
TIM00400		UNTIME DS D		TIM00400
TIM00410		MSG DSECT		TIM00410
TIM00420		HH DS 2C		TIM00420
TIM00430		DS C		TIM00430
TIM00440		MM DS 2C		TIM00440
TIM00450		DS C		TIM00450
TIM00460		SS DS 2C		TIM00460
TIM00470		DS C		TIM00470

TH DS 2C
YY DS C
DDD DS 2C
END DS C
TIME DS 3C

TIM00480
TIM00490
TIM00500
TIM00510
TIM00520
TIM00530


```

*****
*      TITLE 'UNLOAD - DELETE A PREVIOUSLY LOADED MODULE',
*      *****
*      UNLOAD -
*      *****
*      SUBROUTINE TO DELETE A MODULE PREVIOUSLY LOADED
*      *****
*      CALLING SEQUENCE -
*      *****
*      TRUTHVALUE = UNLOAD(FILENAME)
*      *****
*      WHERE FILENAME IS THE FILENAME OF MODULE TO BE DELETED.
*      IT IS UP TO EIGHT CHARACTERS IN LENGTH
*      *****
*      TRUTHVALUE IS TRUE IF DELETE WAS SUCCESSFUL AND FALSE IF
*      DELETE WAS UNSUCCESSFUL
*      *****
*      USES THE OS/VS1 SUPERVISOR 'DELETE' MACRO.  FOR FURTHER INFO,
*      SEE THE 'OS/VS1 SUPERVISOR SERVICES AND MACRO
*      INSTRUCTION', GC24-5103
*      *****
*      UNLOAD      IN      CSECT,STORAGE=0,SAVE=SAVEAREA
*      L            1,0(1)  PICK UP PARM LIST
*      MVC          NAME(8),0(1)  GET LOAD MODULE NAME
*      DELETE       EPLOC=NAME    UNLOAD MODULE (EPLOC = ENTRY POINT LOC)
*      CL           15,'F'4'      COMPARE RETURN CODE TO 4 (COULDN'T FIND)
*      BE          BOMB
*      OUT          OUT          STORAGE=0,SAVE=SAVEAREA,RETC=1,RETREG=0  RETURN 'TRUE'
*      OUT          OUT          STORAGE=0,SAVE=SAVEAREA,RETC=0,RETREG=0  RETURN 'FALSE'
*      *****
*      *** DATA CONSTANT AREA
*      SAVEAREA DS 18F
*      NAME      DC CL8' ,
*      DC        8X'FF'
*      END
*****
UNL00010
UNL00020
UNL00030
UNL00040
UNL00050
UNL00060
UNL00070
UNL00080
UNL00090
UNL00100
UNL00110
UNL00120
UNL00130
UNL00140
UNL00150
UNL00160
UNL00170
UNL00180
UNL00190
UNL00200
UNL00210
UNL00220
UNL00230
UNL00240
UNL00250
UNL00260
UNL00270
UNL00280
UNL00290
UNL00300
UNL00310
UNL00320
UNL00330
UNL00340
UNL00350
UNL00360

```



```

*****
*          CMSCB MACRO - SIMULATED OS CONTROL BLOCK
*
*          USED TO MAP A FILE CONTROL BLOCK
*
*****
MACRO
CMSCB
PUSH PRINT
AIF ('&SYSPARM' NE 'SUP').ACCO1
PRINT OFF,NOGEN
.ACCHO1 ANOP
*****
*          SIMULATED OS CONTROL BLOCKS
*
FCBSECT DSECT
FCBINIT DS OX -
FCBOPCB EQU X'08' -
FCBPERM EQU X'04' -
FCBBATCH EQU X'02' -
FCBCATML EQU X'01' -
FCBOS EQU X'10'
FCBDOSL EQU X'20'
FCBNEXT DS A -
FCBPROC DS A -
FCBDD DS CL8 -
FCBOP DS CL8 -
IHAJFCB DS OD -
JFCBDSNM DS OX -
FCBTAPID DS OX -
FCBDSNAM DS CL8 -
FCBDSTYP DS CL8 -
FCBPRPU EQU FCBSTYP+4 -
FCBTBSP DS OX
FCBDSMD DS CL2 -
FCBITEM DS H -
FCBBUFF DS F -
FCBBYTE DS F -
FCBFORM DS CL2 -
FCBCOUT DS H -
FCBREAD DS F -
FCBDEV DS X -
FCBDUM EQU O -
FCBPTR EQU 4 -
FCBRDR EQU 8 -
FCBCON EQU 12 -

INTERESTING TIDBITS
OPEN ACQUIRED THIS CMS BLOCK
PERMANENT CONTROL BLOCK
SPECIAL BATCH DATA SET
CONCATENATED MACLIB DATA SET
FCB FOR OS FORMATTED DISK
CONCATENATED DOSLIB DATA SET
AL3(NEXT CMSCB)
A(SPECIAL PROCESSING ROUTINE)
DATA DEFINITION NAME
CMS OPERATION
*** JOB FILE CONTROL BLOCK ***
44 BYTES, DATA SET NAME
TAPE IDENTIFICATION
DATA SET NAME
DATA SET TYPE
PRINTER/PUNCH COMMAND LIST
2 BYTES, TAPE BACKSPACE COUNT @VAO4853
DATA SET MODE
ITEM IDENTIFICATION NUMBER
A(INPUT-OUTPUT BUFFER)
DATA COUNT
FILE FORMAT: FIXED/VARIABLE RECORDS
RECORDS PER CMS PHYSICAL BLOCK
N'BYTES ACTUALLY READ
DEVICE TYPE CODE
DUMMY DEVICE
PRINTER
READER
CONSOLE TERMINAL

```


FCBTAP	EQU	16 -	TAPE	CMS00480
FCBDSK	EQU	20 -	DISK	CMS00490
FCBPCH	EQU	24 -	PUNCH	CMS00500
FCBCRT	DS	28 -	CRT	CMS00510
FCBMODE	DS	X -	MODE: 1,2,3,4,5	CMS00520
FCBXTENT	DS	H -	NUMBER OF ITEMS IN EXTENT	CMS00530
FCBRECL	DS	H -	DCB LRECL AT OPEN TIME	CMS00540
IOBIOFLG	DS	X -	I/O FLAGS	CMS00550
FCBDCBCT	DS	X -	NO. OF DCB'S USING THIS FCB	CMS00560
FCBMEMBR	DS	2F	OS PDS MEMBER NAME	CMS00570
FCBOSFST	DS	F	POINTER TO OS FST	CMS00580
FCBOSDSN	DS	F	POINTER TO OS DSNAME BLOCK	CMS00590
FCBR13	DS	F -	SAVEAREA VECTOR R13	CMS00600
FCBKKEYS	DS	A -	A(DDS IN-CORE KEY TABLE)	CMS00610
FCBPDS	DS	A -	A(PDS IN-CORE DIRECTORY)	CMS00620
JFCBMSK	DS	8X -	VARIOUS MASK BITS	CMS00630
JFCBCRODT	DS	3C -	DATA SET CREATION DATE (YDD)	CMS00640
JFCBXPOT	DS	3C -	DATA SET EXPIRATION DATE (YDD)	CMS00650
JFCBIND1	DS	X -	INDICATOR ONE	CMS00660
JFCBIND2	DS	X -	INDICATOR TWO	CMS00670
JFCBUNFO	DS	X -	NUMBER OF BUFFERS	CMS00680
JFCBFTEK	DS	OX -	BUFFERING TECHNIQUE	CMS00690
JFCBFALN	DS	X -	BUFFER ALIGNMENT	CMS00700
JFCBUFL	DS	H -	BUFFER LENGTH	CMS00710
JFCEROPT	DS	X -	ERROR OPTION	CMS00720
JFCKEYLE	DS	X -	KEYLENGTH	CMS00730
	DS	X -	---NOT USED---	CMS00740
JFCCLIMCT	DS	3X -	BDAM SEARCH LIMIT	CMS00750
FCBDSORG	DS	OX -	DATA SET ORGANIZATION	CMS00760
JFCDSORG	DS	2X -		CMS00770
FCBRECFCM	DS	OX -	RECORD FORMAT	CMS00780
JFCRECFCM	DS	X -		CMS00790
JFCOPTCD	DS	X -	OPTION CODES	CMS00800
FCBBLKSZ	DS	OH -	BLOCK SIZE	CMS00810
JFCBLKSI	DS	H -		CMS00820
FCBLRECL	DS	OH -	LOGICAL RECORD LENGTH	CMS00830
JFCLRECL	DS	H -		CMS00840
FCBIOSW	DS	X -	I/O OPERATION INDICATOR	CMS00850
FCBCLOSE	EQU	X'80' -	DURING "CLOSE"	CMS00860
FCBCLEAV	EQU	X'40' -	DISP = LEAVE DURING CLOSE	CMS00870
FCBPROCC	EQU	X'20' -	GOTO FCBPROC DURING CLOSE	CMS00880
FCBPROCO	EQU	X'10' -	GOTO FCBPROC DURING OPEN	CMS00890
FCBCASE	EQU	X'08' -	ON=LOWER CASE CONSOLE I/O	CMS00900
FCBPVMB	EQU	X'04' -	PUT-MOVE-VAR-BLK	CMS00910
FCBIOWR	EQU	X'02' -	WRITE/PUT	CMS00920
FCBIORD	EQU	X'01' -	READ/GET	CMS00930
FCBIOSW2	DS	1X -	I/O OPERATION INDICATORS	CMS00940

FCBMVPS	EQU	X'01' -	SW FOR MOVEFILE WITH PDS OPTION	CMS00950
FCBMV EQU	X'02' -	MOVE PDS SWITCH FOR FIND	@VA03059	CMS00960
FCBMVFIL EQU	X'08' -	MOVE FILE IS ACTIVE		CMS00970
DEBLNGTH DS	OX -	L'DEB IN DBLW WORDS		CMS00980
FCBTCLDS EQU	X'40' -	A CLOSE TYPE T WAS DONE	@VA08024	CMS00990
DS	F -	---NOT USED---		CMS01000
IHADEB DS	OD -	*** DATA EXTENT BLOCK ***		CMS01010
DEBTCBAD DS	A -	A(MOVE-MODE USER BUFFER)		CMS01020
SEBSAV DS	F	DYNAMIC SAVE FOR RET ADDR FOR	@VMO2691	CMS01030
*		SEB (OS I/O SIM)		CMS01040
DEBOFLGS DS	4X -	DATA SET STAU5 FLAGS		CMS01050
DEBOPATB DS	4X -	OPEN/CLOSE OPTION BYTE		CMS01060
IOBFLG DS	OX -	(START OF IOBPREFIX FOR NORMAL SCH)		CMS01070
IOB8FLG EQU	O -	DISPLACEMENT OF IOB FLAG IN IOB		CMS01080
IOBOUT EQU	X'40' -	"WRITE,PUT" IN PROGRESS		CMS01090
IOBIN EQU	X'20' -	"READ,GET" IN PROGRESS		CMS01100
IOBUPD EQU	X'10' -	"QSAM PUTX" IN PROGRESS		CMS01110
IOBNXTAD DS	A -	A(NEXT BUFFER TO BE USED)		CMS01120
IOBECB DS	F -	ECB FOR QSAM NORMAL SCHEDULING		CMS01130
IHAIOB DS	OF -	*** INPUT/OUTPUT BLOCK ***		CMS01140
DEDEBID DS	OX -	DEB IDENTIFICATION		CMS01150
DEDCBAD DS	A -	A(DATA CONTROL BLOCK)		CMS01160
IOBECBCC DS	OX -	ECB COMPLETION CODE		CMS01170
IOBECBC EQU	12 -	DISPLACEMENT OF ECB CODE IN IOB		CMS01180
IOBECBPT EQU	12 -	DISPLACEMENT OF ECB PTR IN IOB		CMS01190
IOBECBPT DS	A -	A(EVENT CONTROL BLOCK)		CMS01200
IOBFLAG3 DS	OX -	I/O ERROR FLAG		CMS01210
IOBBCSW EQU	16 -	DISPLACEMENT OF CSW IN IOB		CMS01220
IOBCSW DS	8X -	LAST CCW STORED(I.E., RESIDUAL COUNT)		CMS01230
IOBSTART DS	A -	X'ID-NEXT BUFFER',AL3(INITIAL BUFFER)		CMS01240
IOBDCBPT DS	A -	A(DATA CONTROL BLOCK)		CMS01250
IOBEND DS	OX -	END-OF-INPUT/OUTPUT BLOCK		CMS01260
FCBEND DS	OD -	END-OF FCB,JFCB,DEB,IOB BLOCKS		CMS01270
FCBENSIZ EQU	(*-FCBSECT)/8	- SIZE OF FCB ENTRY, DOUBLEWORDS		CMS01280
SPACE 3				CMS01290
ORG	FCBDSTYP+4			CMS01300
FCBIOOUT DS	CL8 -	SPECIAL I/O COMMAND LIST		CMS01310
FCBIOBUF DS	A -	A(DATA BUFFER)		CMS01320
FCBCONCR DS	C -	CONSOLE COLOR CODE		CMS01330
FCBCONMS DS	X -	CONSOLE MISCELLANEOUS INFO		CMS01340
FCBIOCNT DS	H -	L'DATA BUFFER		CMS01350
SPACE 3				CMS01360
*				CMS01370
*				CMS01380
IHADECB DSECT				CMS01390
DECSDECB DS	F -	EVENT CONTROL BLOCK		CMS01400
				CMS01410

DECTYPE	DS	H -	TYPE OF I/O REQUEST	CMS01420
DECBRD	EQU	X'80' -	READ SF	CMS01430
DECBWR	EQU	X'20' -	WRITE SF	CMS01440
DECLNGTH	DS	H -	LENGTH OF KEY & DATA	CMS01450
DEDCBAD	DS	A -	V(DATA CONTROL BLOCK)	CMS01460
DECAREA	DS	A -	V(KEY & DATA, BUFFER)	CMS01470
DECIOBPT	DS	A -	V(IOB)	CMS01480
*		BDAM EXTENSION		CMS01490
DECKYADR	DS	A -	V(KEY)	CMS01500
DECRECPT	DS	A -	V(BLOCK REFERENCE FIELD)	CMS01510
		SPACE 3		CMS01520
*		SOME FREQUENTLY USED EQUATES		CMS01530
*				CMS01540
	DDNAM	EQU	FILETYPE = DATA SET NAME	CMS01550
BLK	EQU	X'10' -	RECFM=BLOCKED RECORDS	CMS01560
BS	EQU	X'20' -	MACRF=BSAM	CMS01570
DA	EQU	X'20' -	DSORG=DIRECT ACCESS	CMS01580
FXD	EQU	X'80' -	RECFM=FIXED LENGTH RECORDS	CMS01590
IS	EQU	X'80' -	DSORG=INDEXED SEQUENTIAL	CMS01600
LOC	EQU	X'08' -	MACRF=LOCATE MODE	CMS01610
MOV	EQU	X'10' -	MACRF=MOVE MODE	CMS01620
PS	EQU	X'40' -	DSORG=PHYSICAL SEQUENTIAL	CMS01630
POU	EQU	X'03' -	DSORG=PARTITIONED UNMOVEABLE	CMS01640
PO	EQU	X'02' -	DSORG=PARTITIONED ORGANIZATION	CMS01650
PREVIOUS	EQU	X'80' -	OFLGS=PREVIOUS I/O OPERATION	CMS01660
QS	EQU	X'40' -	MACRF=QSAM	CMS01670
UND	EQU	X'CO' -	RECFM=UNDEFIN	CMS01680
VAR	EQU	X'40' -	RECFM=VARIABLE LENGTH RECORDS	CMS01690
	EJECT			CMS01700
	POP	PRINT		CMS01710
	MEND			CMS01720
				CMS01730


```

*****
*
*      IN MACRO
*
*      USED TO SET UP THE ENTRY CONVENTION FOR AN
*      ASSEMBLY LANGUAGE PROGRAM
*
*****
*
*      MACRO
*      IN      &CSECT,&STORAGE=1024,&DSECT=DSA,&SAVE=
*      AIF     ('&CSECT' EQ 'ENTRY').ENTRY
*      AIF     ('&CSECT' EQ '').NOCSECT
*
*      CSECT
*      SAVE    (14,12),,&LABEL
*      AGO     .AWAY
*
*      ANOP
*      ENTRY  &LABEL
*
*      ANOP
*      .NOCSECT
*      &LABEL
*      .AWAY
*
*      LR      12,15
*      USING  &LABEL,12
*      AIF     ('&STORAGE' EQ 'O').NODSA
*      GETMAIN R,LV=&STORAGE
*      ST      1,8(13)
*      ST      13,4(1)
*      LR      1,13
*      L       13,8(13)
*      USING  &DSECT,13
*      L       0,20(1)
*      L       14,12(1)
*      L       15,16(1)
*      L       1,24(1)
*
*      MEXIT
*
*      .NODSA
*      ANOP
*      AIF     ('&SAVE' EQ '').NOSAVE
*      LA      15,&SAVE
*      ST      13,4(15)
*      ST      15,8(13)
*      LR      13,15
*
*      .NOSAVE
*      ANOP
*      MEND
*****
IN 00010
IN 00020
IN 00030
IN 00040
IN 00050
IN 00060
IN 00070
IN 00080
IN 00090
IN 00100
IN 00110
IN 00120
IN 00130
IN 00140
IN 00150
IN 00160
IN 00170
IN 00180
IN 00190
IN 00200
IN 00210
IN 00220
IN 00230
IN 00240
IN 00250
IN 00260
IN 00270
IN 00280
IN 00290
IN 00300
IN 00310
IN 00320
IN 00330
IN 00340
IN 00350
IN 00360
IN 00370
IN 00380
IN 00390
IN 00400
IN 00410
IN 00420

```



```

*****
*      OUT MACRO
*
*      USED TO SET UP THE RETURN CONVENTION FOR AN
*      ASSEMBLY LANGUAGE PROGRAM
*
*****
MACRO
&LABEL      OUT      &STORAGE=1024,&RETC=0,&RETREG=15,&SAVE=
              GBLB    &REGINIT
              LCLC    &LBL,&SAVEREG
              LCLA    &OFFSET
              &LBL    SETC  ' &LABEL '
              AIF     (' &STORAGE' EQ '0').NODSA
              LR      1,13
              L        13,4(13)
              FREEMAIN R,LV=&STORAGE,A=(1)
              &LBL    SETC  ' '
              AGO     .RETURN
              .NODSA  ANOP
              AIF     (' &SAVE' EQ ' ').RETURN
              &LBL    L      13,4(13)
              &LBL    SETC  ' '
              .RETURN ANOP
              &LBL    LA      &RETREG,&RETC
              AIF     (' &REGINIT' EQ 'SET').REGSIN
              &OFFSET SETA    (&RETREG+2)*4+12
              AIF     (&RETREG LT 13).LOW
              &OFFSET SETA    (&RETREG-13)*4+8
              .LOW    ANOP
              &SAVEREG SETC  ' &OFFSET '
              AGO     .ON
              .REGSIN ANOP
              &SAVEREG SETC  'SR'.&RETREG'
              .ON    ANOP
              &SAVEREG SETC  ' &SAVEREG'. '(13)'
              ST      &RETREG,&SAVEREG
              LM      14,12,12(13)
              BR      14
              LTORG
              MEND
*****
OUT00010
OUT00020
OUT00030
OUT00040
OUT00050
OUT00060
OUT00070
OUT00080
OUT00090
OUT00100
OUT00110
OUT00120
OUT00130
OUT00140
OUT00150
OUT00160
OUT00170
OUT00180
OUT00190
OUT00200
OUT00210
OUT00220
OUT00230
OUT00240
OUT00250
OUT00260
OUT00270
OUT00280
OUT00290
OUT00300
OUT00310
OUT00320
OUT00330
OUT00340
OUT00350
OUT00360
OUT00370
OUT00380
OUT00390
OUT00400
OUT00410
OUT00420

```


APPENDIX D

DEX DATABASE EDITING ROUTINES


```

C *****
C LOGICAL FUNCTION DBCMPR(RCODE)
C +-----+
C
C LOGICAL FUNCTION DBCMPR CAN BE CALLED TO COMPRESS THE
C DATABASE AFTER A NUMBER OF DELETES.
C
C THE ARGUMENT RCODE IS RETURNED TO INDICATE THE REASON FOR
C THE LOGICAL VALUE OF DBCMPR UPON RETURN.
C
C RCODE = 0 --> THE COMPRESS WAS SUCCESSFUL.
C RCODE = 1 --> THE COMPRESS FAILED BECAUSE THERE WERE
C NO DELETED NODES.
C
C +-----+
C DATABASE COMMON
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C DIMENSION
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C INTEGER
C DELNOD
C
C +++ INITIALIZE CONSTANTS
C
C COUNT=0
C
C SEQUENCE THROUGH THE DATABASE LOOKING FOR DELETED ENTRIES
C DELETED ENTRIES HAVE NODTYP = -9, WHEN A DELETED NODE IS
C FOUND COMPRESS UNTILL THE NEXT DELETED NODE, THEN INCREMENT
C COUNT AND CONTINUE COMPRESSING BY COUNT. ADJUST POINTERS
C ACCORDINGLY.
C
C +++ BEGIN SEARCH LOOP
C DO 1000 I=1,MAXNOD
C +++ IF NODTYP(I)=0 THE REST OF THE DB NODES ARE EMPTY
C IF (NODTYP(I) .EQ. 0) GO TO 6000
C IF (NODTYP(I) .NE. -9) GO TO 1000

```



```

C      *** A DELETED ENTRY WAS FOUND, MOVE THE REST UP
      DELNOD=I
      COUNT=COUNT+1
      GO TO 2000
1000 CONTINUE
C      *** END OF SEARCH LOOP
C      *** IF YOU REACH THIS POINT THERE WERE NO DELETED NODES
      GO TO 7777
2000 CONTINUE
C      *** BEGIN COMPRESS LOOP
      DO 3000 J=DELNOD,MAXNOD
        IF (NODTYP(J) .EQ. 0) GO TO 6000
        IF (J .EQ. MAXNOD) GO TO 5000
        IDENT(J,1)=IDENT(J+COUNT,1)
        IDENT(J,2)=IDENT(J+COUNT,2)
        NODTYP(J)=NODTYP(J+COUNT)
      ***
      *** LINK AND LINKF ARE MOVED UP BY COUNT. IF LINK(J)=0
      *** THEN THIS IS THE LAST NODE OR ONLY NODE IN A CHAIN.
      *** IF LINK(J) IS NOT EQUAL TO ZERO THEN THE NODE TO WHICH DBCC00670
      *** LINK(J) POINTS MUST HAVE ITS LINKF(LINK(J)) REDUCED BY DBCC00680
      *** COUNT.
      ***
      *** IF LINKF(J) IS NEGATIVE THEN IT POINTS TO THE BUCKET
      *** WHICH IS POINTING TO THIS NODE. SO THE BUCKET ENTRY
      *** MUST BE ADJUSTED TO POINT TO THE NEW NODE POSITION J.
      ***
      LINK(J)=LINK(J+COUNT)
      LINKF(J)=LINKF(J+COUNT)
      IF (LINK(J) .NE. 0) LINKF(LINK(J))=LINKF(LINK(J))-COUNT
      IF (LINKF(J) .LT. 0) BUCKET(IABS(LINKF(J)))=J
      CMTPTR(J)=CMTPTR(J+COUNT)
      DATUM(J)=DATUM(J+COUNT)
      ***
      *** IF THE CMTPTR IS NOT ZERO THERE IS A COMMENT, SO
      *** UPDATE THE POINTER BACK TO THE NODE FROM THE STORAGE
      *** AREA. IF THE NODTYP IS EQUAL TO 3 THEN THERE IS
      *** AN ARRAY STORED, SO UPDATE THE POINTER BACK TO THE
      *** NODE FROM THE ARRAY STORAGE AREA.
      ***
      IF (CMTPTR(J) .EQ. 0) GO TO 2400
      IPOINT = CMTPTR(J)
      CALL WRITEC(J,IPOINT+2,1,ECSPTR(1),ECSLEN(1))
      IF (NODTYP(J) .NE. 3) GO TO 2500
      IPOINT = DATUM(J)
2400

```



```

C      CALL WRITEC(J,IPOINT+2,1,ECSPTR(1),ECSLEN(1))
C
C      +++
C      +++ IF NODTYP(J)=-9 AFTER COMPRESS THEN THE NEXT DELETED
C      +++ ENTRY HAS BEEN FOUND.  UPDATE THE NODE NUMBER OF THE
C      +++ DELNOD AND EXIT THE LOOP.  COUNT WILL BE INCREMENTED
C      +++ AND THE COMPRESS LOOP ENTERED WITH THE NEW DELNOD,
C      +++ AND COUNT.
C      +++
C
C      2500      IF (NODTYP(J) .EQ. -9) DELNOD=J
C      3000      IF (NODTYP(J) .EQ. -9) GO TO 4000
C      4000      CONTINUE
C      5000      +++ END OF COMPRESS LOOP
C      6000      +++ MOVE ZEROS INTO THE LAST DATABASE NODE
C      77777      IDENT(MAXNOD,1)=BLANK
C      80000      IDENT(MAXNOD,2)=BLANK
C      90000      NODTYP(MAXNOD)=0
C      100000      DATUM(MAXNOD)=0
C      110000      LINK(MAXNOD)=0
C      120000      LINKF(MAXNOD)=0
C      130000      CMTPTR(MAXNOD)=0
C      140000      CONTINUE
C      150000      +++ COMPRESS WAS SUCCESSFUL RCODE = 0
C      160000      DBCMPR=.TRUE.
C      170000      DELCNT=0
C      180000      NAVAIL=NAVAIL-COUNT
C      190000      RCODE=0
C      200000      GO TO 99999
C      210000      77777 CONTINUE
C      220000      +++ THERE WERE NO DELETED NODES RETURN DBCMPR=.FALSE.
C      230000      +++ AND RCODE = 1
C      240000      DBCMPR=.FALSE.
C      250000      RCODE=1
C      260000      GO TO 99999
C      270000      99999 CONTINUE
C      280000      RETURN
C      290000      END

```

DBC00950
 DBC00960
 DBC00970
 DBC00980
 DBC00990
 DBC01000
 DBC01010
 DBC01020
 DBC01030
 DBC01040
 DBC01050
 DBC01060
 DBC01070
 DBC01080
 DBC01090
 DBC01100
 DBC01110
 DBC01120
 DBC01130
 DBC01140
 DBC01150
 DBC01160
 DBC01170
 DBC01180
 DBC01190
 DBC01200
 DBC01210
 DBC01220
 DBC01230
 DBC01240
 DBC01250
 DBC01260
 DBC01270
 DBC01280
 DBC01290
 DBC01300
 DBC01310
 DBC01320
 DBC01330
 DBC01340
 DBC01350
 DBC01360


```

4      INTEGER
1      MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDV, HELPDV,
2      OUTDEV, INPDEV,
3      MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFL,
4      OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL, NFWD
C...  FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY. ....
      DIMENSION
1      MSGSFL(11), USEFIL(11), INFOFL(11),
2      DBSFIL(11), NEWSFL(11), HELPFL(11),
3      OUTFIL(11), INPFIL(11),
4      LOADFL(11), NOT1FL(11), NOT2FL(11)
      COMMON /DBCOM/
1      TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
2      CMTPTTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
3      INTTYP, RELTYP, ARRTYP,
4      DBCLSD, DBACTV, FILEDB
      INTEGER
1      TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
2      CMTPTTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
3      INTTYP, RELTYP, ARRTYP
      LOGICAL
1      DBCLSD, DBACTV, FILEDB
      DIMENSION
1      TITLE(16), BUCKET(32), IDENT(200,2), NODTYP(200),
2      DATUM(200), LINK(200), LINKF(200), CMTPTTR(200)
      INTEGER TYPE, DXMENU, COMMAN, ARSIZE, ONE, ISPAC1, ISPAC2
      INTEGER WDSPA1, WDSPA2, WDSPA3
      LOGICAL LOG, DXNBRI, DXGNAM
      DIMENSION NAME(2), MITEM1(20), MITEM2(6)
      DIMENSION MENNM1(2), MENNM2(2)
      DIMENSION ISPAC1(16), ISPAC2(16), RSPAC3(200)
      DATA WDSPA1 /16/, WDSPA2 /16/, WDSPA3 /200/
      DATA NAMSUB /4HEDIT/
      DATA ONE /1/
      DATA MENNM1 /4HDBED, 4HCMDS/
      DATA NITEM1 /10/
      DATA MITEM1 /4HCREA, 4HTE, 4HSTOR, 4HE, 4HDELE, 4HTE,
+      4HCOMM, 4HENT, 4HEXPL, 4HAIN, 4HPRIN, 4HT,
+      4HDUMP, 4H, 4HSET-, 4HTITL, 4HGET-, 4HTITL,
+      4HDOONE, 4H /
      DATA MENNM2 /4HDB-T, 4HYPES/
      DATA NITEM2 /3/
      DATA MITEM2 /4HINTE, 4HGER, 4HREAL, 4H, 4HARRA, 4HY-RL/
      IF (DBCLSD) GO TO 1111
      IF (FILEDB) CALL DXMSGZ(NAMSUB,5)
      IF (.NOT.(FILEDB)) CALL DXMSGZ(NAMSUB,6)
      CALL DXFNPR(DBSFIL)
      CALL DBEDTG(ISPA1, WDSPA1)

```



```

C.... CONTINUE
1000
C...   *** REQUEST A EDIT-DB COMMAND
      CALL DXMSGF(NAMSUB,1,ISPAC1,WDSPA1)
      COMMAN=DXMENU(MENNM1,NITEM1,MITEM1,ISPAC1)
      IF (DXREQS) GO TO 99999
      IF (COMMAN.GT.6) GO TO 3000
      IF (COMMAN.GT.2) GO TO 2000
      *** REQUEST TYPE OF DATUM
      CALL DXMSGF(NAMSUB,2,ISPAC1,WDSPA1)
      TYPE=DXMENU(MENNM2,NITEM2,MITEM2,ISPAC1)
      IF (DXREQS) GO TO 99999
      IF (TYPE.LT.3) GO TO 2000
      *** FOR ARRAYS REQUEST SIZE
      CALL DXMSGF(NAMSUB,3,ISPAC1,WDSPA1)
      LOG=DXNBRI(ONE,ISPAC2,RPAC3,NEEDUM,ISPAC1,1)
      IF (DXREQS) GO TO 99999
      IF (.NOT.LOG) GO TO 99999
      ARSIZE=ISPAC2(1)

2000      CONTINUE
C...   *** REQUEST NAME OF THE DATUM
      LOG=DXGNAM(NAME)
      IF (DXREQS) GO TO 99999

3000      CONTINUE
      GO TO (3010,3020,3030,3040,3050,
+         3060,3070,3080,3090,3100),COMMAN

3010 CONTINUE
C...   *** CREATE A NODE IN THE DATABASE
      CALL DBEDCR(NAME,TYPE,ARSIZE)
      GO TO 1000

3020 CONTINUE
C...   *** STORE A DATA VALUE IN THE NODE
      CALL DBEDST(NAME,TYPE,ARSIZE,RPAC3,WDSPA3)
      IF (DXREQS) GO TO 99999
      GO TO 1000

3030 CONTINUE
C...   *** DELETE A NODE
      CALL DBEDDL(NAME)
      GO TO 1000

3040 CONTINUE
C...   *** PUT A COMMENT FOR THE NODE INTO THE DATABASE
      CALL DBEDCM(NAME,ISPAC1,WDSPA1,ISPAC2,WDSPA2,NCHCMT)
      IF (DXREQS) GO TO 99999
      GO TO 1000

3050 CONTINUE
C...   *** EXPLAIN THE NODE BY DISPLAYING THE COMMENT
      CALL DBEDEX(NAME,ISPAC1,WDSPA1,NCHCMT)

```


3060	GO TO 1000	DBEO1420
C...	CONTINUE	DBEO1430
	+++ PRINT A SPECIFIED DATUM VALUE	DBEO1440
	CALL DBEDPR(NAME,ARSIZE,RSPAC3,WDSPA3)	DBEO1450
	GO TO 1000	DBEO1460
3070	CONTINUE	DBEO1470
C...	+++ DUMP THE WHOLE DATABASE	DBEO1480
	CALL DBEDDM(ISPAC1,WDSPA1,RSPAC3,WDSPA3)	DBEO1490
	IF (DXREQ5) GO TO 99999	DBEO1500
	GO TO 1000	DBEO1510
3080	CONTINUE	DBEO1520
C...	+++ STORE TITLE IN DATABASE	DBEO1530
	CALL DBEDTS(ISPAC1,WDSPA1,ISPAC2,WDSPA2,NCHTIT)	DBEO1540
	IF (DXREQ5) GO TO 99999	DBEO1550
	GO TO 1000	DBEO1560
3090	CONTINUE	DBEO1570
C...	+++ GET THE TITLE FROM THE DATABASE	DBEO1580
	CALL DBEDTG(ISPAC1,WDSPA1)	DBEO1590
	GO TO 1000	DBEO1600
3100	CONTINUE	DBEO1610
C...	+++ DONE WITH THIS MENU, SO RETURN	DBEO1620
	GO TO 99999	DBEO1630
C...		DBEO1640
11111	CONTINUE	DBEO1650
	CALL DXMSGZ(NAMSUB,4)	DBEO1660
	GO TO 99999	DBEO1670
99999	CONTINUE	DBEO1680
	RETURN	DBEO1690
	END	DBEO1700


```

C *****
C SUBROUTINE DBEDCR(NAME,TYPE,ARSize)
C +-----+
C DBEDCR -
C
C CREATE A NODE IN THE DATABASE FOR THE GIVEN VARIABLE
C NAME. TYPE IS 1,2, OR 3 FOR INTEGER, REAL, REAL-ARRAY.
C IF TYPE = 3 THEN ARSize IS THE SIZE OF THE ARRAY.
C
C +-----+
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION DXSC(1),CURMOD(2),NULMSG(1)
C INTEGER TYPE,RCODE,ARSize
C LOGICAL DBVINS,DBCMPR
C DIMENSION NAME(2)
C DATA NAMSUB /4HEDIT/
C
C 1000 IF (DBVINS(NAME,TYPE,ARSize,RCODE)) GO TO 99999
C
C
C
C IF (RCODE.EQ.2) CALL DXMSGC(NAMSUB,7,NAME,NAMLEN,DXNCPW)
C
C IF (RCODE.EQ.3) GO TO 11111
C GO TO 99999
C
C 11111 IF (DBCMPR(RCODE)) GO TO 1000
C CALL DXMSGZ(NAMSUB,8)
C GO TO 99999
C
C 99999 CONTINUE
C RETURN
C END

```

DBE01710
DBE01720
DBE01730
DBE01740
DBE01750
DBE01760
DBE01770
DBE01780
DBE01790
DBE01800
DBE01810
DBE01820
DBE01830
DBE01840
DBE01850
DBE01860
DBE01870
DBE01880
DBE01890
DBE01900
DBE01910
DBE01920
DBE01930
DBE01940
DBE01950
DBE01960
DBE01970
DBE01980
DBE01990
DBE02000
DBE02010
DBE02020
DBE02030
DBE02040
DBE02050
DBE02060
DBE02070
DBE02080
DBE02090
DBE02100
DBE02110
DBE02120
DBE02130
DBE02140


```

C *****
C SUBROUTINE DBEDST(NAME,TYPE,ARSIZE,RARRAY,WDARSP)
C +-----+
C
C DBEDST -
C
C STORE A VALUE OF GIVEN TYPE IN THE NODE PREVIOUSLY
C CREATED FOR THE VARIABLE NAME.
C
C ARSIZE IS THE SIZE OF THE ARRAY TO BE STORED.
C
C RARRAY IS A TEMPORARY IN WHICH THE ARRAY WILL BE STORED
C UNTIL THE ELEMENTS ARE ENTERED INTO THE ECS STORAGE AREA.
C
C WDARSP IS THE DIMENSION OF RARRAY.
C
C +-----+
C *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C COMMON /DBCOM/
C 1 TITLE,BUCKET,IDENT,NODTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE,BUCKET,IDENT,NODTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER TYPE,STYPE,ONE,TWO,RCODE,ARSIZE,DBNSKR,WDARSP
C LOGICAL DXNBRI,RPUT,IPUT,APUT,LOG,DBXEC5

```



```

DIMENSION NAME(2), RARRAY(WDARSP), IVALUE(1), RVALUE(1), IDUM(1),
+      DUM(1)
EQUIVALENCE ( IVALUE(1), RVALUE(1) )
DATA NAMSUB /4HEDIT/
DATA      ONE /1/, TWO/2/
C
C...      *** SEARCH FOR THE NODE USING THE HASHED VALUE AS AN INDEX
C      DBE02620
      DBE02630
      DBE02640
      DBE02650
      DBE02660
      DBE02670
      DBE02680
      DBE02690
      DBE02700
      DBE02710
      DBE02720
      DBE02730
      DBE02740
      DBE02750
      DBE02760
      DBE02770
      DBE02780
      DBE02790
      DBE02800
      DBE02810
      DBE02820
      DBE02830
      DBE02840
      DBE02850
      DBE02860
      DBE02870
      DBE02880
      DBE02890
      DBE02900
      DBE02910
      DBE02920
      DBE02930
      DBE02940
      DBE02950
      DBE02960
      DBE02970
      DBE02980
      DBE02990
      DBE03000
      DBE03010
      DBE03020
      DBE03030
      DBE03040
      DBE03050
      DBE03060
      DBE03070
      DBE03080

C...      NODE=DBNSKR(NAME, STYPE, IDUM, RCODE)
C...      IF (NODE.EQ.O) GO TO 1111
C...      IF ( TYPE.NE.IABS(STYPE) ) GO TO 2222
C...      GO TO (1000,2000,3000), TYPE
C...      CONTINUE
C...      *** STORE AN INTEGER VALUE
      LOG=DXNBRI(ONE, IVALUE, DUM, NEED, NULMSG, ONE)
      IF (DXREQS) GO TO 9999
      IF ( .NOT. LOG) GO TO 9999
      LOG=IPUT(NAME, IVALUE, RCODE)
      GO TO 9999
C...      CONTINUE
C...      *** STORE A REAL VALUE
      LOG=DXNBRI(ONE, IDUM, RVALUE, NEED, NULMSG, TWO)
      IF (DXREQS) GO TO 9999
      IF ( .NOT. LOG) GO TO 9999
      LOG=RPUT(NAME, RVALUE, RCODE)
      GO TO 9999
C...      CONTINUE
C...      *** STORE A REAL ARRAY
      N=ARSIZE
      IF (N.GT. MAXARR) GO TO 3333
      LOG=DXNBRI(N, IDUM, RARRAY, NEED, NULMSG, TWO)
      IF (DXREQS) GO TO 9999
      IF (NEED.EQ.N) GO TO 9999
      N=N-NEED
      LOG=APUT(NAME, RARRAY, N, NSTORD, RCODE)
      IF (RCODE.EQ.4.OR.RCODE.EQ.5) GO TO 4444
      GO TO 9999
C...      CONTINUE
C...      *** NODE DOES NOT EXIST
      CALL DXMSGC(NAMSUB, 9, NAME, NAMLEN, DXNCPW)
      GO TO 9999
C...      CONTINUE
C...      *** WRONG TYPE
      CALL DXMSGZ(NAMSUB, 10)
      GO TO 9999

```



```

33333 CONTINUE
C...      +-+ ARRAY TOO BIG TO BE STORED
          CALL DXMSGZ(NAMSUB,11)
          GO TO 99999
44444 CONTINUE
C...      +-+ ARRAY PUT DID NOT TAKE PLACE SO EXPAND THE ECS AREA
          LOG=DBXEC(S>IDUM)
          IF (LOG) GO TO 3000
          +-+ COULD NOT EXPAND SO ANNOUNCE AND RETURN
          CALL DXMSGI(NAMSUB,17,NSTORD)
          GO TO 99999
99999 CONTINUE
          RETURN
          END
DBE03090
DBE03100
DBE03110
DBE03120
DBE03130
DBE03140
DBE03150
DBE03160
DBE03170
DBE03180
DBE03190
DBE03200
DBE03210
DBE03220

```



```

C *****
C SUBROUTINE DBEDDL (NAME)
C +-----+
C DBEDDL -
C
C      DELETE A NODE FROM THE DATABASE BY MAKING IT UNAVAILABLE.
C      THIS IS DONE BY FLAGGING NODTYP AS -9
C +-----+
C
C      *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C      *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C      COMMON /DEXCOM/
C      1  DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C      2  TERSE, ECHOMD, DXSC, CPWDXS, MDSC, CURMOD,
C      3  NULMSG, DXNCPW, NAMLEN,
C      4  CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE, GCNTRL
C
C      INTEGER
C      1  DXSC, CPWDXS, MDSC, CURMOD, NULMSG, DXNCPW,
C      2  NAMLEN, CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE
C
C      LOGICAL
C      1  DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C      2  TERSE, ECHOMD, GCNTRL
C
C      DIMENSION
C      1  DXSC(1), CURMOD(2), NULMSG(1)
C
C      INTEGER RCODE
C      LOGICAL DBVDEL
C      DIMENSION NAME(2)
C      DATA NAMSUB /4HEDIT/
C
C      +++ INVOKE THE LOGICAL FUNCTION DBVDEL TO DELETE THE NODE
C      IF (DBVDEL(NAME,RCODE)) GO TO 99999
C      +++ UNABLE TO FIND NAME TO DELETE, ANNOUNCE.
C      CALL DXMSGC(NAMSUB,9,NAME,NAMLEN,DXNCPW)
C      GO TO 99999
99999 CONTINUE
      RETURN
      END

```

```

DBEO3230
DBEO3240
DBEO3250
DBEO3260
DBEO3270
DBEO3280
DBEO3290
DBEO3300
DBEO3310
DBEO3320
DBEO3330
DBEO3340
DBEO3350
DBEO3360
DBEO3370
DBEO3380
DBEO3390
DBEO3400
DBEO3410
DBEO3420
DBEO3430
DBEO3440
DBEO3450
DBEO3460
DBEO3470
DBEO3480
DBEO3490
DBEO3500
DBEO3510
DBEO3520
DBEO3530
DBEO3540
DBEO3550
DBEO3560
DBEO3570
DBEO3580
DBEO3590
DBEO3600

```



```

C...      GO TO 99999
11111 CONTINUE
C...      +++ NODE DOES NOT EXIST
          CALL DXMSGC(NAMSUB,9,NAME,NAMLEN,DXNCPW)
          GO TO 99999
22222 CONTINUE
C...      +++ NO MORE SPACE FOR COMMENTS, EXPAND THE ECS AREA
          LOG=DBXECS(IDUM)
          IF (LOG) GO TO 1000
          +++ COULD NOT EXPAND ANNOUNCE AND RETURN
          CALL DXMSGZ(NAMSUB,13)
          GO TO 99999
99999 CONTINUE
          RETURN
          END
DBE04080
DBE04090
DBE04100
DBE04110
DBE04120
DBE04130
DBE04140
DBE04150
DBE04160
DBE04170
DBE04180
DBE04190
DBE04200
DBE04210
DBE04220
DBE04230

```



```

C *****
C SUBROUTINE DBEDEX(NAME,CMTBUF,WDSBUF,NCHCMT)
C +-----+
C
C DBEDEX -
C
C EXPLAIN THE DATABASE ENTRY BY DISPLAYING ITS COMMENT.
C
C NAME IS THE NAME OF THE DATUM.
C
C CMTBUF IS THE ARRAY BUFFER WHICH THE COMMENT IS PLACED IN
C NCHCMT IS THE NUMBER OF CHARACTERS IN THE COMMENT.
C +-----+
C
C *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1
C 2
C LOGICAL
C 1
C 2
C DIMENSION DXSC(1),CURMOD(2),NULMSG(1)
C INTEGER CMTBUF,RCODE,WDSBUF
C LOGICAL LOG,CMTGET
C DIMENSION CMTBUF(WDSBUF),NAME(2)
C DATA NAMSUB /4HEDIT/
C
C +++ GET THE COMMENT FROM THE DATABASE.
C
C LOG=CMTGET(NAME,CMTBUF,RCODE)
C IF (RCODE.EQ.2) GO TO 1111
C IF (RCODE.EQ.3) GO TO 2222
C CALL DXMSGN(CMTBUF,NCHCMT)
C GO TO 9999
C
C 11111 CONTINUE
C
C +++ NODE DOES NOT EXIST
C
C CALL DXMSGC(NAMSUB.9,NAME,NAMLEN,DXNCPW)
C GO TO 9999
C
C 22222 CONTINUE
C
C +++ NO COMMENT STORED
C
C CALL DXMSGZ(NAMSUB.14)

```


GO TO 99999
99999 CONTINUE
RETURN
END

DBEO4710
DBEO4720
DBEO4730
DBEO4740


```

C *****
C SUBROUTINE DBEDPR(NAME,ARSIZE,ARSPAC,WDARSP)
C +-----+
C
C DBEDPR -
C
C PRINT THE VALUE OF A DATABASE ENTRY AT THE TERMINAL.
C
C NAME IS THE NAME OF THE DATUM.
C
C ARSIZE IS THE SIZE OF THE ARRAY, ARSPAC IS A TEMPORARY
C ARRAY IN WHICH TO STORE THE ARRAY, AND WDARSP IS THE
C DIMENSION OF ARSPAC.
C +-----+
C
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE, LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE ,ECHOMD,DXSC ,CPWDXS,MDSC ,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC ,CPWDXS,MDSC ,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE, LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE ,ECHOMD,GCNTRL
C
C DIMENSION
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C REAL RDATUM
C INTEGER STYPE,RCODE,IDATUM,DBNSKR,UNDEF,UNDLLEN,UNDCPW
C INTEGER ARSIZE,WDARSP

```



```

DIMENSION NAME(2),UNDEF(3),ARSPAC(WDARSP)
EQUIVALENCE (IDATUM,RDATUM)
DATA NAMSUB /4HEDIT/
DATA UNDEF /4HUNDE,4HFINE,4HDI /
DATA UNLEN /10/
C...      *** SEARCH THE DATABASE FOR THE NODE
NODE=DBNSKR(NAME,STYPE,IDUM,RCODE)
IF (NODE.EQ.0) GO TO 11111
      *** EXTRACT THE VALUE
      IDATUM=DATUM(NODE)
      CALL DXMSGC(NAMSUB,15,NAME,NAMLEN,DXNCPW)
      IF (STYPE.LT.0) GO TO 2222
      *** DETERMINE THE TYPE
      GO TO (1000,2000,3000),STYPE
      CONTINUE
C...      *** PRINT INTEGER DATUM
      CALL DXMSGI(NAMSUB,15,IDATUM)
      GO TO 9999
      CONTINUE
2000      *** PRINT REAL DATUM
C...      CALL DXMSGR(NAMSUB,15,RDATUM)
      GO TO 9999
      CONTINUE
3000      *** PRINT REAL-ARRAY
C...      CALL DBEDAP(NAME,IDATUM,WDARSP,ARSPAC,WDARSP)
      GO TO 9999
C....
11111 CONTINUE
      *** NAMED DATUM DOES NOT EXIST
C...      CALL DXMSGC(NAMSUB,9,NAME,NAMLEN,DXNCPW)
      GO TO 9999
2222 CONTINUE
C...      *** DATUM IS UNDEFINED
      CALL DXMSGC(NAMSUB,16,UNDEF,UNLEN,DXNCPW)
      GO TO 9999
9999 CONTINUE
      RETURN
      END

```

DBE05220
 DBE05230
 DBE05240
 DBE05250
 DBE05260
 DBE05270
 DBE05280
 DBE05290
 DBE05300
 DBE05310
 DBE05320
 DBE05330
 DBE05340
 DBE05350
 DBE05360
 DBE05370
 DBE05380
 DBE05390
 DBE05400
 DBE05410
 DBE05420
 DBE05430
 DBE05440
 DBE05450
 DBE05460
 DBE05470
 DBE05480
 DBE05490
 DBE05500
 DBE05510
 DBE05520
 DBE05530
 DBE05540
 DBE05550
 DBE05560
 DBE05570
 DBE05580
 DBE05590


```

C *****
C SUBROUTINE DBEDAP(NAME,NARRAY,NGET,RARRAY,WDARSP)
C +-----+
C
C DBEDAP -
C
C PRINT THE CONTENTS OF AN ARRAY FROM THE DATABASE.
C
C NAME IS THE NAME OF THE DATUM.
C
C NARRAY IS THE POINTER TO THE LOCATION OF THE ARRAY IN
C THE ECS STORAGE AREA. NGET IS THE NUMBER TO GET.
C
C RARRAY IS THE ARRAY IN WHICH TO PUT THE VALUES, AND
C WDARSP IS THE DIMENSION OF RARRAY.
C +-----+
C
C *****
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSCL,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSCL,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C 1 DXSC(1),CURMOD(2),NULMSG(1)
C
C *****
C *I/O DEVICES AND FILES
C
C COMMON /DEXFIL/
C 1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELDPV,
C 2 OUTDEV,INPDEV,NFWDSC,
C 3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL
C
C INTEGER
C 1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELDPV,
C 2 OUTDEV,INPDEV,
C 3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL,NFWDSC
C
C FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY....
C DIMENSION
C 1 MSGSFL(11),USEFIL(11),INFOFL(11),
C DBSFIL(11),NEWSFL(11),HELPL(11),

```



```

3      OUTFL(11) , INPFL(11) ,
2      LOADFL(11) , NOT1FL(11) , NOT2FL(11)
C...  *THE MESSAGE HANDLING STORE AND PARAMETERS

      COMMON /DEXMSG/
1      DXCHAR, MDCHAR, TRIMCH, MAXMSG, MAXWDS, MXPOOL,
2      SUBCNT, SUBCOD, MSGPTR, PVERB, LVERB, PTERSE,
3      LTERSE, LSTRNG, VPOSIT, TPOSIT, MSPOOL

      INTEGER
1      DXCHAR, MDCHAR, TRIMCH, MAXMSG, MAXWDS, MXPOOL,
2      SUBCNT, SUBCOD, MSGPTR, PVERB, LVERB, PTERSE,
3      LTERSE, LSTRNG, VPOSIT, TPOSIT, MSPOOL

      DIMENSION
5      SUBCOD( 40), MSGPTR(102), PVERB( 102),
6      LVERB( 102), PTERSE(102), LTERSE(102),
7      LSTRNG(102), VPOSIT(102), TPOSIT(102),
      MSPOOL(950), TRIMCH(1)

      INTEGER START, WDARSP
      LOGICAL DBNARD, LOG
      DIMENSION NAME(2), RARRAY(WDARSP)
      DATA START /1/

C...  *** READ THE ARRAY FROM ECS STORAGE
      LOG=DBNARD(NARRAY,RARRAY,START,NGET,NSTORD)
      CALL DXGMSG
      WRITE (OUTDEV,00001) DXCHAR,NAME,NSTORD,NARRAY
      JSTEP=5
      DO 2000 I=1,NSTORD,ISTEP
        JBEG=1
        JEND=MINO(I+ISTEP-1,NSTORD)
        DO 1000 J=JBEG,JEND,JSTEP
          KBEG=J
          KEND=MINO(J+JSTEP-1,JEND)
          CALL DXGMSG
          WRITE (OUTDEV,00002) DXCHAR, (RARRAY(K),K=KBEG,KEND)
          *** UPDATE GRAPHICS

C...  CONTINUE
1000  CALL DXWAIT
      CALL DXUGRF(ERASE)

2000  CONTINUE
      CALL DXUGRF(UPDATE)
      RETURN
00001 FORMAT (1H /1H ,A1,6HNAME: ,2A4,9H, LENGTH=,I3,3X,1H(,I5,1H)/)
00002 FORMAT (1H ,A1,5E14.5)
      END

```



```

C *****
C SUBROUTINE DBEDDM(CMTBUF,CMTWDS,RARRAY,RARWDS)
C +-----+
C
C DBEDDM -
C
C DUMP THE CONTENTS OF THE DATABASE TO THE TERMINAL
C TO BE DISPLAYED.
C
C CMTBUF IS AN ARRAY IN WHICH TO STORE COMMENTS TEMPORARILY
C CMTWDS IS THE NUMBER OF COMMENT WORDS.
C
C RARRAY IS AN ARRAY IN WHICH TO STORE THE ARRAY TEMPORARILY
C RARWDS IS THE NUMBER OF WORDS IN THE ARRAY.
C +-----+
C
C *****
C *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1
C 2
C LOGICAL
C 1
C 2
C DIMENSION
C DXSC(1),CURMOD(2),NULMSG(1)
C
C *****
C *I/O DEVICES AND FILES
C
C COMMON /DEXFIL/
C 1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELPDV,
C 2 OUTDEV,INPDEV,NFWDs,
C 3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL
C
C INTEGER
C 1
C 2
C 3
C 4
C
C FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY
C DIMENSION
C 1 MSGSFL(11),USEFIL(11),INFOFL(11),
C 2 DBSFIL(11),NEWSFL(11),HELPL(11),
C 3 OUTFIL(11),INPFIL(11),

```



```

C....      2      LOADFL(11) ,NOT1FL(11) ,NOT2FL(11)
              *THE MESSAGE HANDLING STORE AND PARAMETERS
              COMMON /DEXMSG/
1              DXCHAR,MDCCHAR,TRIMCH,MAXMSG,MAXWDS,MXPOOL,
2              SUBCNT,SUBCOD,MSGPTR,PVERB,LVERB,PTERSE,
3              LTERSE,LSTRNG,VPOSIT,TPOSIT,MSPOOL
              INTEGER
1              DXCHAR,MDCCHAR,TRIMCH,MAXMSG,MAXWDS,MXPOOL,
2              SUBCNT,SUBCOD,MSGPTR,PVERB,LVERB,PTERSE,
3              LTERSE,LSTRNG,VPOSIT,TPOSIT,MSPOOL
              DIMENSION
5              SUBCOD( 40),MSGPTR(102),PVERB (102),
6              LVERB (102),PTERSE(102),LTERSE(102),
7              LSTRNG(102),VPOSIT(102),TPOSIT(102),
              MSPool(950),TRIMCH(1)
              COMMON /DBCOR/
1              TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1              CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2              INTTYP,RELTYP,ARRTYP,
3              DBCLSD,DBACTV,FILEDB
              INTEGER
1              TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1              CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2              INTTYP,RELTYP,ARRTYP
              LOGICAL
3              DBCLSD,DBACTV,FILEDB
              DIMENSION
1              DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
              INTEGER      BKTPTTR,TYPE,CMTBUF,CMTWDS,RARWDS,UTYPE,CLASS
              LOGICAL      NOCOMM,UNDEFD,DBNARD,DBNCRD,LOG
              DIMENSION NAME(2),CMTBUF(CMTWDS),RARRAY(RARWDS),CLASS(3)
              EQUIVALENCE (IDATUM,RDATUM)
              DATA CLASS /4H (1),4H (R),4H (A)/
              CALL DXUGRF(ERASE)
              CALL DBEDTG(CMTBUF ,CMTWDS)
              CALL DXGMSG
              WRITE (OUTDEV,00001) DXCHAR
              C....      *** SCAN THE BUCKET FOR NODE ENTRIES
              DO 7000 BKTPTTR=1,32
                  NODE=BUCKET(BKTPTTR)
                  CONTINUE
                  IF (NODE.EQ.O) GO TO 6000
                  *** IF A NODE IS NOT ZERO DUMP IT
                  NAME(1)=IDENT(NODE,1)
                  NAME(2)=IDENT(NODE,2)
                  NOCOMM=(CMTPTR(NODE) .EQ.O)
                  IF (.NOT.NOCOMM) LOG=DBNCRD(CMTPTR(NODE),CMTBUF)
                  TYPE=NODTYP(NODE)

```



```

2100 UNDEFD=(TYPE,LT,1)
C... IF (UNDEFD) GO TO 3000
      IDATUM=DATUM(NODE)
      GO TO (2100,2200,2300),TYPE
      CONTINUE
      +++ DUMP THE INTEGER DATUM
      IF (NOCOMM) GO TO 2110
      WRITE (OUTDEV,00002)
      NAME,CLASS(TYPE),IDATUM,CMTBUF
      GO TO 4000
      CONTINUE
2110 WRITE (OUTDEV,00002) NAME,CLASS(TYPE),IDATUM
      GO TO 4000
      CONTINUE
      +++ DUMP THE REAL DATUM
      IF (NOCOMM) GO TO 2210
      WRITE (OUTDEV,00003)
      NAME,CLASS(TYPE),RDATUM,CMTBUF
      GO TO 4000
      CONTINUE
2210 WRITE (OUTDEV,00003) NAME,CLASS(TYPE),RDATUM
      GO TO 4000
      CONTINUE
      +++ INDICATE AN ARRAY POINTER
      IF (NOCOMM) GO TO 2310
      WRITE (OUTDEV,00004)
      NAME,CLASS(TYPE),IDATUM,CMTBUF
      GO TO 4000
      CONTINUE
2310 WRITE (OUTDEV,00004) NAME,CLASS(TYPE),IDATUM
      GO TO 4000
      CONTINUE
      +++ INDICATE THE UNDEFINED ONES
      UTYPE=IABS(TYPE)
      IF (NOCOMM) GO TO 3010
      WRITE (OUTDEV,00005) NAME,CLASS(UTYPE),CMTBUF
      GO TO 4000
      CONTINUE
3010 WRITE (OUTDEV,00005) NAME,CLASS(UTYPE)
      GO TO 4000
      CONTINUE
      NODE=LINK(NODE)
      GO TO 1000
4000 CONTINUE
      CONTINUE
      CONTINUE
6000 CONTINUE
7000 WRITE (OUTDEV,00006)
      +++ DUMP THE ARRAYS
C...

```



```

DO 9000 NODE=1,MAXNOD
  IF (NODTyp(NODE).NE.3) GO TO 8000
  NAME(1)=IDENT(NODE,1)
  NAME(2)=IDENT(NODE,2)
  NARRAY=DATUM(NODE)
  NGET=RARWDS
  CALL DBEDAP(NAME,NARRAY,NGET,RARRAY,RARWDS)
8000  CONTINUE
9000  CONTINUE
      CALL DXUGRF(UPDATE)
      CALL DXWAIT
99999 CONTINUE
      RETURN
C....
00001 FORMAT (//1H ,A1,2X,4HNAME,4X,4HTYPE,11X,
+ SHVALUE,2X,7HCOMMENT/)
00002 FORMAT (1H ,3X,3A4,I16,2X,16A4)
00003 FORMAT (1H ,3X,3A4,E16.5,2X,16A4)
00004 FORMAT (1H ,3X,3A4,4X,6HARRAY(,I5,1H),2X,16A4)
00005 FORMAT (1H ,3X,3A4,3X,13H**UNDEFINED**,2X,16A4)
00006 FORMAT (1H //11H THE ARRAYS/)
      END

```

```

DBE07910
DBE07920
DBE07930
DBE07940
DBE07950
DBE07960
DBE07970
DBE07980
DBE07990
DBE08000
DBE08010
DBE08020
DBE08030
DBE08040
DBE08050
DBE08060
DBE08070
DBE08080
DBE08090
DBE08100
DBE08110
DBE08120

```



```

C *****
C SUBROUTINE DBEDTS(TITLE,WDSITIT,MSGAG,MSGWDS,NCHTIT)
C +-----+
C
C DBEDTS -
C
C STORE A TITLE IN THE DATABASE.
C
C TITLE IS THE ARRAY WHERE THE DATABASE TITLE IS STORED
C TEMPORARILY, WDSITIT IS ITS DIMENSION.
C
C MSGAG IS AN ARRAY IN WHICH TO STORE A MESSAGE, AND
C MSGWDS IS ITS DIMENSION.
C
C NCHTIT IS THE NUMBER OF CHARACTERS IN THE TITLE.
C +-----+
C
C *****
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C DXSC(1),CURMOD(2),NULMSG(1)
C
C INTEGER TITLE,CPWTIT,WDSITIT,MSGAG,MSGWDS
C
C LOGICAL DXCHRI,LOG,DXPRMT,TPUT
C
C DIMENSION TITLE(WDSITIT),MSGAG(MSGWDS)
C
C DATA NAMSUB /4HEDTS/
C
C CALL DXMSGF(NAMSUB,1,MSGAG,MSGWDS)
C
C *** GET THE TITLE FROM THE USER
C LOG=DXCHRI(NCHTIT,TITLE,WDSITIT,DXNCPW,NEED,MSGAG)
C IF (DXREQS) GO TO 99999
C
C *** PUT THE TITLE IN THE DATABASE
C LOG=TPUT(TITLE)
C
C 99999 CONTINUE
C RETURN
C END

```



```

C *****
C SUBROUTINE DBEDTG(TITLE,WDSTIT)
C +-----+
C
C DBEDTG -
C
C GET THE TITLE OF THE DATABASE.
C
C TITLE IS AN ARRAY IN WHICH TO STORE THE TITLE.
C WDSTIT IS THE NUMBER OF WORDS IN THE TITLE.
C +-----+
C
C *** I/O DEVICES AND FILES ***
C... COMMON /DEXFIL/
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDV, HELPDV,
C 2 OUTDEV, INPDEV, NFWD,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL
C
C INTEGER
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDV, HELPDV,
C 2 OUTDEV, INPDEV,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL, NFWD
C... FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY.....
C DIMENSION
C 1 MSGSFL(11), USEFIL(11), INFOFL(11),
C 2 DBSFIL(11), NEWSFL(11), HELPFL(11),
C 3 OUTFIL(11), INPFIL(11),
C 4 LOADFL(11), NOT1FL(11), NOT2FL(11)
C... *THE MESSAGE HANDLING STORE AND PARAMETERS*
C... COMMON /DEXMSG/
C 1 DXCHAR, MDCHAR, TRIMCH, MAXMSG, MAXWDS, MXPPOOL,
C 2 SUBCNT, SUBCOD, MSGPTR, PVERB, LVERB, PTERSE,
C 3 LTERSE, LSTRNG, VPOSIT, TPOSIT, MSPPOOL
C
C INTEGER
C 1 DXCHAR, MDCHAR, TRIMCH, MAXMSG, MAXWDS, MXPPOOL,
C 2 SUBCNT, SUBCOD, MSGPTR, PVERB, LVERB, PTERSE,
C 3 LTERSE, LSTRNG, VPOSIT, TPOSIT, MSPPOOL
C DIMENSION
C 5 SUBCOD( 40), MSGPTR(102), PVERB (102),
C 6 LVERB (102), PTERSE(102), LTERSE(102),
C 7 LSTRNG(102), VPOSIT(102), TPOSIT(102),
C 8 MSPPOOL(950), TRIMCH(1)
C... INTEGER TITLE, WDSTIT
C... LOGICAL TGET
C... DIMENSION TITLE(WDSTIT)
C... DATA NAMSUB /4HEDTG/

```



```

C....      IF (.NOT.(TGET(TITLE))) GO TO 1111
            CALL DXMSG
C....      ++ GET THE TITLE FROM THE DATABASE USING L.F. TGET
            ++ DISPLAY THE TITLE AT THE TERMINAL
            WRITE (OUTDEV,00001) DXCHAR,TITLE
            GO TO 9999
11111 CONTINUE
C....      ++ NO TITLE STORED
            CALL DXMSGZ(NAMSUB,1)
            GO TO 9999
99999 CONTINUE
            RETURN
00001 FORMAT (1H ,A1,9HTITLE: **,16A4,2H**)
            END
DBE09060
DBE09070
DBE09080
DBE09090
DBE09100
DBE09110
DBE09120
DBE09130
DBE09140
DBE09150
DBE09160
DBE09170
DBE09180
DBE09190

```



```

C *****
C SUBROUTINE DBINIT *****
C +-----+
C DBINIT -
C
C      INITIALIZE THE DATABASE IN MEMORY.  CALLED FOR EACH
C      NEW DATABASE.
C +-----+
C
C      COMMON /DBCOM/
C      1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C      1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT ,
C      2 INTTYP,RELTYP,ARRTYP,
C      3 DBCLSD,DBACTV,FILEDB
C
C      INTEGER
C      1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C      1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C      2 INTTYP,RELTYP,ARRTYP
C
C      LOGICAL
C      3 DBCLSD,DBACTV,FILEDB
C
C      DIMENSION
C      1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200),
C      *BUFFER AND PARSER ARRAYS AND PARAMETERS
C
C      COMMON /DEXPRS/
C      1 INBUFR,BUFLEN,CPWIBU,FLAGST,FLAGLN,FLAGCH,
C      1 LENBUF,CURPOS,NXTPOS,BEGPOS,ENDPOS,INPEND,
C      1 DELCHR,DELNUM,CPWDEL,STRDEL,CPWAQU,DELETE,
C      1 BLANK,
C      1 INTG,REAL,LINTG,LREAL,MENU,MOXY,XY
C
C      INTEGER
C      1 INBUFR,BUFLEN,CPWIBU,FLAGST,FLAGLN,FLAGCH,
C      1 LENBUF,CURPOS,NXTPOS,BEGPOS,ENDPOS,
C      1 DELCHR,DELNUM,CPWDEL,STRDEL,CPWAQU,DELETE,
C      1 BLANK,
C      1 INTG,REAL,LINTG,LREAL,MENU,MOXY,XY
C
C      LOGICAL
C      1 INPEND
C
C      DIMENSION
C      1 INBUFR(133),FLAGST(133),FLAGCH(1),DELCHR(1),
C      STRDEL(1),DELETE(1)
C
C      COMMON /DBDATAIO/
C      1 MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
C
C      INTEGER
C      1 PRECS,ECSPTR,ECSLEN
C
C      DIMENSION
C      1 ECSPTR(2),ECSLEN(2)

```



```

LOGICAL      GETMN,GETFLG
DATA      MAXL /2000/
DATA      NAMSUB /4HINDB/
C...      +++ INITIALIZE THE MAX SIZES
          MAXNOD=200
          MAXARR=200
          NCHTIT=64
          NCHCMT=64
C...      +++ INITIALIZE THE TYPE CODES
          INTTYP=1
          RELTYP=2
          ARRTYP=3
C...      +++ INITIALIZE THE DATABASE VALUES
          NAVAIL = 1
          DELCNT = 0
          DO 1000 I = 1,MAXNOD
            IDENT(I,1)=BLANK
            IDENT(I,2)=BLANK
            NODTYP(I)=0
            LINK(I) = I+1
            LINKF(I) = 0
            CMTPTR(I) = 0
          CONTINUE
          LINK(MAXNOD) = 0
          DO 2000 I=1,32
            BUCKET(I) = 0
          CONTINUE
          DO 3000 I=1,16
            TITLE(I)=0
          CONTINUE
          NEXECS=1
          PRECS=0
          MAXECS=MAXL
C...      +++ GET MAIN STORAGE TO SIMULATE CDC ECS
          GETFLG=GETMN(MAXECS,ECSPTR(1))
          IF (.NOT. GETFLG) GO TO 88888
          ECSLEN(1)=MAXECS
          GO TO 99999
88888      CONTINUE
C...      +++ NOT ENOUGH STORAGE LEFT IN THE USERS VIRTUAL MACHINE.
          CALL DXMSGZ (NAMSUB,1)
99999      CONTINUE
          RETURN
          END
DBE09670
DBE09680
DBE09690
DBE09700
DBE09710
DBE09720
DBE09730
DBE09740
DBE09750
DBE09760
DBE09770
DBE09780
DBE09790
DBE09800
DBE09810
DBE09820
DBE09830
DBE09840
DBE09850
DBE09860
DBE09870
DBE09880
DBE09890
DBE09900
DBE09910
DBE09920
DBE09930
DBE09940
DBE09950
DBE09960
DBE09970
DBE09980
DBE09990
DBE10000
DBE10010
DBE10020
DBE10030
DBE10040
DBE10050
DBE10060
DBE10070
DBE10080
DBE10090
DBE10100

```



```

C *****
C INTEGER FUNCTION DBGETN(RCODE)
C +-----+
C
C DBGETN -
C
C GET THE NUMBER OF THE NEXT AVAILABLE NODE.
C
C RCODE = 0 THERE ARE NODES STILL AVAILABLE
C RCODE = 1 NO MORE NODES AVAILABLE
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER RCODE
C DBGETN=NAVAIL
C... IF (DBGETN.EQ.O) RCODE=1
C... IF (DBGETN.NE.O) RCODE=O
C IF (RCODE.EQ.O) NAVAIL=LINK(DBGETN)
C GO TO 99999
99999 CONTINUE
C RETURN
C END

```

```

DBE 10110
DBE 10120
DBE 10130
DBE 10140
DBE 10150
DBE 10160
DBE 10170
DBE 10180
DBE 10190
DBE 10200
DBE 10210
DBE 10220
DBE 10230
DBE 10240
DBE 10250
DBE 10260
DBE 10270
DBE 10280
DBE 10290
DBE 10300
DBE 10310
DBE 10320
DBE 10330
DBE 10340
DBE 10350
DBE 10360
DBE 10370
DBE 10380
DBE 10390
DBE 10400
DBE 10410
DBE 10420
DBE 10430
DBE 10440
DBE 10450
DBE 10460
DBE 10470
DBE 10480

```



```

C *****
C LOGICAL FUNCTION DBNDIN(DNAME,TYPE,IDATUM,RCODE)
C +-----+
C DBNDIN -
C
C ENTER THE DATUM AND TYPE FOR A NODE INTO THE DATABASE.
C
C DNAME IS THE NAME OF THE DATUM.
C
C RCODE IS A RETURN CODE
C
C RCODE = 0 DATUM SUCCESSFULLY ENTERED
C RCODE = 1 NO OPEN DATABASE
C RCODE = 2 NAMED DATUM DOES NOT EXIST
C RCODE = 3 TYPE DISAGREEMENT
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER DNAME,TYPE,IDATUM,RCODE,DBNSKR
C INTEGER PRNODE,STYPE
C DIMENSION DNAME(2)
C IF (DBCLSD) GO TO 1111
C   +++ SEARCH FOR THE NODE
C   NODE=DBNSKR(DNAME,STYPE,PRNODE,RCODE)
C   DBNDIN=(NODE.NE.O)
C   IF (.NOT.DBNDIN) GO TO 2222
C   IF (IABS(STYPE).NE.TYPE) GO TO 3333
C   +++ IF FOUND ENTER THE DATUM AND TYPE
C   NODTYP(NODE)=TYPE
C   DATUM(NODE)=IDATUM
C   RCODE=0
C   GO TO 9999
C...
C...

```



```

11111 CONTINUE
C...      +++ NO OPEN DATA BASE
          RCODE=1
          GO TO 99999
22222 CONTINUE
C...      +++ NAMED DATUM DOES NOT EXIST
          RCODE=2
          GO TO 99999
33333 CONTINUE
C...      +++ TYPE DISAGREEMENT
          RCODE=3
          GO TO 99999
99999 CONTINUE
          RETURN
          END

```

```

DBE 10960
DBE 10970
DBE 10980
DBE 10990
DBE 11000
DBE 11010
DBE 11020
DBE 11030
DBE 11040
DBE 11050
DBE 11060
DBE 11070
DBE 11080
DBE 11090
DBE 11100

```



```

C *****
C LOGICAL FUNCTION DBNDRD(DNAME,TYPE, IDATUM,RCODE)
C +-----+
C DBNDRD -
C
C READ THE DATUM FOR A NODE IN THE DATABASE.
C
C DNAME IS THE DATUM NAME, AND TYPE IS 1, 2, OR 3 FOR
C INTEGER, REAL, OR REAL-ARRAY.
C
C THE DATUM IS PUT IN IDATUM.
C
C RCODE IS A RETURN CODE.
C
C RCODE = 0 THE READ WAS SUCCESSFUL
C RCODE = 1 NO OPEN DATABASE
C RCODE = 2 DNAME NOT IN DATABASE
C RCODE = 3 WRONG TYPE
C RCODE = 4 NO DATUM STORED YET
C
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 BKTLNK,CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,
C 1 DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 BKTLNK,CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,
C 1 DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),BKTLNK(200),
C 1 CMTPTR(200)
C
C INTEGER DNAME,TYPE,IDATUM,RCODE,DBNSKR
C INTEGER PRNODE,STYPE
C DIMENSION DNAME(2)
C IF (DBCLSD) GO TO 11111
C IDATUM=0
C
C +++ SEARCH THE DATABASE FOR THE NODE
C NODE=DBNSKR(DNAME,STYPE,PRNODE,RCODE)
C ...

```



```

DBNDRD=(NODE.NE.O)
IF (.NOT.DBNDRD) GO TO 22222
IF (IABS(STYPE).NE.TYPE) GO TO 33333
  +++ READ THE DATUM INTO IDATUM
  IDATUM=DATUM(NODE)
  IF (STYPE.LT.O) GO TO 44444
  RCODE=O
  GO TO 99999
11111 CONTINUE
  +++ NO OPEN DATA BASE
C... DBNDRD=.FALSE.
      RCODE=1
      GO TO 99999
22222 CONTINUE
  +++ NOT IN DATA BASE
C... RCODE=2
      GO TO 99999
33333 CONTINUE
  +++ WRONG TYPE
C... RCODE=3
      GO TO 99999
44444 CONTINUE
  +++ NO DATUM STORED
C... RCODE=4
      GO TO 99999
99999 CONTINUE
      RETURN
      END
DBE 11580
DBE 11590
DBE 11600
DBE 11610
DBE 11620
DBE 11630
DBE 11640
DBE 11650
DBE 11660
DBE 11670
DBE 11680
DBE 11690
DBE 11700
DBE 11710
DBE 11720
DBE 11730
DBE 11740
DBE 11750
DBE 11760
DBE 11770
DBE 11780
DBE 11790
DBE 11800
DBE 11810
DBE 11820
DBE 11830
DBE 11840
DBE 11850

```



```

1000 CONTINUE      +++ IF NODE = 0 THE DATUM WASN'T FOUND
C...              IF (NODE.EQ.0) GO TO 3000
C...              +++ CHECK FOR A MATCH BETWEEN NAME AND IDENT
                  IF ((IDENT(NODE,1).EQ.DNAME(1))
+                  .AND.(IDENT(NODE,2).EQ.DNAME(2))) GO TO 2000
                  PRNODE=NODE
                  RCODE=0
                  NODE=LINK(PRNODE)
                  GO TO 1000
2000 CONTINUE
C...              +++ DATUM WAS FOUND
                  TYPE = NODTYP(NODE)
                  GO TO 1111
3000 CONTINUE
C...              +++ DATUM WAS NOT FOUND
                  PRNODE=HASH
                  RCODE=1
                  GO TO 1111
1111 CONTINUE
                  DBNSKR=NODE
                  GO TO 9999
9999 CONTINUE
                  RETURN
                  END
DBE 12330
DBE 12340
DBE 12350
DBE 12360
DBE 12370
DBE 12380
DBE 12390
DBE 12400
DBE 12410
DBE 12420
DBE 12430
DBE 12440
DBE 12450
DBE 12460
DBE 12470
DBE 12480
DBE 12490
DBE 12500
DBE 12510
DBE 12520
DBE 12530
DBE 12540
DBE 12550
DBE 12560
DBE 12570

```



```

C *****
C SUBROUTINE DBOPUT(NAMEDF,NEWBAS,FILNAM,RCODE)
C +-----+
C DBOPUT -
C
C DATABASE OPEN UTILITY - ROUTINE FOR OPENING A DATABASE
C IN MEMORY.
C
C RCODE IS A RETURN CODE:
C
C RCODE = 0 SUCCESSFUL
C RCODE = 1 AN OPEN DATABASE ALREADY EXISTS
C RCODE = 2 FAILED TO LOAD THE DATABASE
C
C +-----+
C
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDESC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDESC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C 1 DXSC(1),CURMOD(2),NULMSG(1)
C
C *I/O DEVICES AND FILES
C
C COMMON /DEXFIL/
C 1 MSGSDV,USEDEV,INFODV,DSDEV,NEWSDV,HELDPV,
C 2 OUTDEV,INPDEV,NFWDs,
C 3 MSGSFL,USEFIL,INFOFL,DSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL
C
C INTEGER
C 1 MSGSDV,USEDEV,INFODV,DSDEV,NEWSDV,HELDPV,
C 2 OUTDEV,INPDEV,
C 3 MSGSFL,USEFIL,INFOFL,DSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL,NFWDs
C
C FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY....
C DIMENSION
C 1 MSGSFL(11),USEFIL(11),INFOFL(11),
C DBSFIL(11),NEWSFL(11),HELPL(11),

```



```

3      OUTFIL(11) ,INPFIL(11) ,NOT2FL(11)
2      LOADFL(11) ,NOT1FL(11) ,NOT2FL(11)
COMMON /DBCOM/
1      TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT ,
2      INTTYP,RELTYP,ARRTYP,
3      DBCLSD,DBACTV,FILEDB
INTEGER
1      TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT ,
2      INTTYP,RELTYP,ARRTYP
LOGICAL
3      DBCLSD,DBACTV,FILEDB
DIMENSION
1      TITLE(16) ,BUCKET(32),IDENT(200,2),NODTYP(200) ,
      DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
INTEGER DXMENU,FILNAM,RCODE
INTEGER CODBEN(20),XCODE,TEMP(20)
LOGICAL DBLOAD,DXCKDV,NAMEDF,NEWBAS,MAKNEW,DXFNEQ,DBSAME,DXPRMT
DIMENSION MSG(16),MENNAM(2),ITEMS(4)
DATA MENNAM /4HDXYE,4HS-NO/
DATA NITEMS /2/
DATA ITEMS /4HYES ,4H ,4HNO ,4H /
DATA MSGWDS/16/,NAMSUB/4HOPUT/
C
IF (NAMEDF) GO TO 600
IF (.NOT.DBCLSD) GO TO 700
GO TO 900
600 CONTINUE
IF (DBCLSD) GO TO 3000
GO TO 1111
700 CONTINUE
C...
+++ ANNOUNCE THE NAME OF THE OPEN DATABASE
WRITE(OUTDEV,00001)
00001 FORMAT(' THE OPEN DATABASE IS ')
CALL DXFNPR(DBSFIL)
CALL DXFNMV(DBSFIL,TEMP)
C...
+++ ASK IF DBSFIL SHALL BE USED
CALL DXMSGF(NAMSUB,2,MSG,MSGWDS)
ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
IF (DXREQS) GO TO 99999
IF (ITEM.EQ.1) GO TO 750
IF (ITEM.EQ.2) GO TO 800
750 CONTINUE
C...
+++ ASK IF WE WANT TO SAVE BEFORE WE USE
CALL STRPAK(MSG,MSGWDS,4H< ,51HDO YOU WANT TO SAVE OPEN DATABASESE
1 BEFORE USING IT?<)
ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)

```



```

IF (DXREQS) GO TO 99999
IF (ITEM.EQ.2) GO TO 99999
CALL DBCLUT(.FALSE.,CCDBFN,XCODE)
IF (DXREQS) GO TO 99999
IF (XCODE.GT.O) GO TO 1111
GO TO 7000
800 CONTINUE
CALL DBCLUT(.FALSE.,CCDBFN,XCODE)
IF (DXREQS) GO TO 99999
IF (XCODE .GT.O) GO TO 1111
IF (NAMEDF) GO TO 3000
GO TO 2000
900 CONTINUE
IF (NAMEDF) GO TO 3000
1000 CONTINUE
IF (.NOT.(DBACTV)) GO TO 2000
+++ ANNOUNCE THE NAME OF THE ACTIVE DATABASE
CALL DXMSGZ(NAMSUB,1)
CALL DXFNPR(DBSFIL)
CALL DXFNMV(DBSFIL,TEMP)
+++ ASK IF DBSFIL SHALL BE USED
CALL DXMSGF(NAMSUB,2,MSG,MSGWDS)
ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
IF (DXREQS) GO TO 99999
IF (ITEM.EQ.1) GO TO 7000
2000 CONTINUE
+++ ASK FOR A NAME
IF (DXPRMT(O)) CALL DXMSGZ(NAMSUB,3)
CALL DXGFNM(DBSFIL)
IF (.NOT.DXREQS) GO TO 2100
IF (DBACTV) CALL DXFNMV(TEMP,DBSFIL)
GO TO 99999
2100 CONTINUE
IF (NEWBAS) GO TO 5000
+++ IS A SAVED BASE TO BE USED
CALL DXMSGF(NAMSUB,4,MSG,MSGWDS)
ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
IF (.NOT.DXREQS) GO TO 2200
IF (DBACTV) CALL DXFNMV(TEMP,DBSFIL)
GO TO 99999
2200 CONTINUE
GO TO (6000,5000),ITEM
3000 CONTINUE
IF (.NOT.(NEWBAS)) GO TO 4000
CALL DXFNMV(FILNAM,DBSFIL)
GO TO 5000
4000 CONTINUE

```

DBE 13520
 DBE 13530
 DBE 13540
 DBE 13550
 DBE 13560
 DBE 13570
 DBE 13580
 DBE 13590
 DBE 13600
 DBE 13610
 DBE 13620
 DBE 13630
 DBE 13640
 DBE 13650
 DBE 13660
 DBE 13670
 DBE 13680
 DBE 13690
 DBE 13700
 DBE 13710
 DBE 13720
 DBE 13730
 DBE 13740
 DBE 13750
 DBE 13760
 DBE 13770
 DBE 13780
 DBE 13790
 DBE 13800
 DBE 13810
 DBE 13820
 DBE 13830
 DBE 13840
 DBE 13850
 DBE 13860
 DBE 13870
 DBE 13880
 DBE 13890
 DBE 13900
 DBE 13910
 DBE 13920
 DBE 13930
 DBE 13940
 DBE 13950
 DBE 13960
 DBE 13970
 DBE 13980


```

5000      IF ((DBACTV).AND.(DXFNEQ(FILNAM,DBSFIL))) GO TO 7000
          CALL DXFNMV(FILNAM,DBSFIL)
          GO TO 6000
          CONTINUE
          MAKNEW=DXCKDV(DBSDEV,DBSFIL)
          IF (.NOT.MAKNEW) GO TO 2222
          CALL DBINIT
          FILEDB=.FALSE.
          GO TO 7000
          CONTINUE
          FILEDB=DBLOAD(RCODE)
          IF (FILEDB) CALL DXMSGZ(NAMSUB,7)
          IF (FILEDB) GO TO 7000
          IF (NAMEDF) GO TO 2222
          +++ WE ARE ABLE TO TRY AGAIN, ANNOUNCE
          +++ FAILURE
          CALL DXMSGZ(NAMSUB,5)
          CALL DXFNPR(DBSFIL)
          +++ ASK IF TO TRY AGAIN
          CALL DXMSGF(NAMSUB,6,MSG,MSGWDS)
          ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
          IF (.NOT.DXREQS) GO TO 6300
          IF (DBACTV) CALL DXFNMV(TEMP,DBSFIL)
          GO TO 9999
6300      CONTINUE
          GO TO (2000,2222),ITEM
          CONTINUE
          +++ SUCCESS
          DBCLSD=.FALSE.
          DBACTV=.TRUE.
          RCODE =0
          GO TO 9999
11111     CONTINUE
          +++ AN OPEN DATABASE EXISTS, CAN NOT OPEN ANOTHER
          RCODE=1
          IF (NAMEDF) GO TO 9999
          CALL DXFNPR(DBSFIL)
          CALL DXMSGZ(NAMSUB,8)
          GO TO 9999
22222     CONTINUE
          +++ LOADING FAILURE
          DBACTV=.FALSE.
          RCODE =2
          GO TO 9999
99999     CONTINUE

```

```

DBE 13990
DBE 14000
DBE 14010
DBE 14020
DBE 14030
DBE 14040
DBE 14050
DBE 14060
DBE 14070
DBE 14080
DBE 14090
DBE 14100
DBE 14110
DBE 14120
DBE 14130
DBE 14140
DBE 14150
DBE 14160
DBE 14170
DBE 14180
DBE 14190
DBE 14200
DBE 14210
DBE 14220
DBE 14230
DBE 14240
DBE 14250
DBE 14260
DBE 14270
DBE 14280
DBE 14290
DBE 14300
DBE 14310
DBE 14320
DBE 14330
DBE 14340
DBE 14350
DBE 14360
DBE 14370
DBE 14380
DBE 14390
DBE 14400
DBE 14410
DBE 14420
DBE 14430
DBE 14440

```



```

      RETURN
      END
C
C *****
C SUBROUTINE DBCLUT(NAMEDF,FILNAM,RCODE)
C +-----+
C
C DBCLUT -
C
C DATABASE CLOSE UTILITY - ROUTINE FOR CLOSING AN OPEN
C DATABASE.
C
C RCODE IS A RETURN CODE;
C
C RCODE = 0 DATABASE WAS SUCCESSFULLY CLOSED.
C RCODE = 1 NO ACTIVE DATABASE.
C RCODE = 2 DATABASE COULD NOT BE SAVED WITH THE
C GIVEN NAME.
C +-----+
C
C *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C 2 TERSE, ECHOMD, DXSC, CPWDXS, MDSC, CURMOD,
C 3 NULMSG, DXNCPW, NAMLEN,
C 4 CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE, GCNTRL
C
C INTEGER
C 1 DXSC, CPWDXS, MDSC, CURMOD, NULMSG, DXNCPW,
C 2 NAMLEN, CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE
C
C LOGICAL
C 1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C 2 TERSE, ECHOMD, GCNTRL
C
C DIMENSION
C DXSC(1), CURMOD(2), NULMSG(1)
C
C *** I/O DEVICES AND FILES
C COMMON /DEXFIL/
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDEV, HELPDV,
C 2 OUTDEV, INPDEV, NFWD,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFIL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL
C
C INTEGER
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDEV, HELPDV,
C 2 OUTDEV, INPDEV,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFIL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL, NFWD
C
C *** FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY. ....

```



```

DIMENSION      MSGSFL(11) , USEFIL(11) , INFOFL(11) ,
1 DBSFIL(11) , NEWSFL(11) , HELPFIL(11) ,
3 OUTFIL(11) , INPFIL(11) ,
2 LOADFL(11) , NOT1FL(11) , NOT2FL(11)
COMMON /DBCOM/
1 TITLE , BUCKET,IDENT , NODTYP,DATUM , LINK , LINKF ,
1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2 INTTYP,RELTYP,ARRTYP,
3 DBCLSD,DBACTV,FILEDB
INTEGER
1 TITLE , BUCKET,IDENT , NODTYP,DATUM , LINK , LINKF ,
1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2 INTTYP,RELTYP,ARRTYP
LOGICAL
3 DBCLSD,DBACTV,FILEDB
DIMENSION
1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
   DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
INTEGER FILNAM,RCODE,DXMENU
INTEGER NAME(1),TEMP(20)
LOGICAL NAMEDF,DBSAVE,DXPRMT
DIMENSION FILNAM(987)
DIMENSION MSG(16),MENNAM(2),ITEMS(4)
DATA MENNAM /4HDXYE,4HS-NO/
DATA MSGWDS/16/,NITEMS /2/
DATA ITEMS /4HYES ,4H ,4HNO ,4H /
DATA NAMSUB/4HCLUT/
DATA NAME /4HOPUT/

C.... IF (DBCLSD) GO TO 11111
CALL DXFNMV(DBSFIL,TEMP)
IF (.NOT.(NAMEDF)) GO TO 1000
CALL DXFNMV(FILNAM,DBSFIL)
GO TO 4000

1000 CONTINUE
C.... +++ ANNOUNCE THE NAME OF THE OPEN DATABASE
CALL DXMSGZ(NAMSUB,1)
CALL DXFNPR(DBSFIL)
C.... +++ SHOULD DATABASE BE SAVED\
CALL DXMSGF(NAMSUB,2,MSG,MSGWDS)
ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
IF (DXREQ) GO TO 99999
IF (ITEM.EQ.1) GO TO 2000
DBACTV=.FALSE.
GO TO 6000

2000 CONTINUE
C.... +++ SHOULD IT BE SAVED WITH A NEW FILE NAME\
CALL DXMSGF(NAMSUB,3,MSG,MSGWDS)

```



```

ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
IF (DXREQS) GO TO 99999
GO TO (3000,4000),ITEM
3000 CONTINUE
C...   *** GET THE NEW FILE NAME
        CALL DXGFNM(DBSFIL)
        IF (DXREQS) GO TO 99999
            GO TO 4000
4000 CONTINUE
C...   *** SAVE THE FILE
        DBACTV=DBSAVE(DBSFIL)
        FILEDB=DBACTV
        IF (DBACTV) GO TO 6000
            IF (.NOT.(NAMEDF)) GO TO 5000
        *** THE DATABASE COULD NOT BE SAVED AS NAMED IN
        *** FILNAM
            DBACTV= .TRUE.
            CALL DXFNMV(TEMP,DBSFIL)
            RCODE=2
            GO TO 99999
C...
5000 CONTINUE
C...   *** THE DATABASE COULD NOT BE SAVED WITH THE
        *** INTERACTIVELY OBTAINED NAME. TRY AGAIN BUT
        *** PROMPT FIRST
            CALL DXMSGZ(NAMSUB,5)
            CALL DXFNPR(DBSFIL)
            DBACTV= .TRUE.
            CALL DXFNMV(TEMP,DBSFIL)
            CALL DXMSGF(NAME,6,MSG,MSGWDS)
            ITEM=DXMENU(MENNAM,NITEMS,ITEMS,MSG)
            IF (DXREQS) GO TO 99999
                GO TO (1000,5100),ITEM
5100 CONTINUE
        RCODE=2
        GO TO 99999
6000 CONTINUE
        RCODE=0
        GO TO 7000
7000 CONTINUE
        DBCLSD= .TRUE.
        GO TO 99999
11111 CONTINUE
C...   *** THERE IS NO ACTIVE DATABASE
        IF (.NOT.(NAMEDF)) CALL DXMSGZ(NAMSUB,4)
        RCODE=1
        GO TO 99999
DBE 15390
DBE 15400
DBE 15410
DBE 15420
DBE 15430
DBE 15440
DBE 15450
DBE 15460
DBE 15470
DBE 15480
DBE 15490
DBE 15500
DBE 15510
DBE 15520
DBE 15530
DBE 15540
DBE 15550
DBE 15560
DBE 15570
DBE 15580
DBE 15590
DBE 15600
DBE 15610
DBE 15620
DBE 15630
DBE 15640
DBE 15650
DBE 15660
DBE 15670
DBE 15680
DBE 15690
DBE 15700
DBE 15710
DBE 15720
DBE 15730
DBE 15740
DBE 15750
DBE 15760
DBE 15770
DBE 15780
DBE 15790
DBE 15800
DBE 15810
DBE 15820
DBE 15830
DBE 15840
DBE 15850

```


99999 CONTINUE
RETURN
END

DBE 15860
DBE 15870
DBE 15880


```

C C *****
C C LOGICAL FUNCTION DBOPEN(NAMEDF,NEWBAS,FILNAM,RCODE)
C C +-----+
C C DBOPEN -
C C
C C ROUTINE TO CALL DBOPUT TO OPEN A DATABASE
C C
C C RCODE AS RETURNED FROM DBOPUT
C C +-----+
C C
C C INTEGER FILNAM,RCODE
C C LOGICAL NAMEDF,NEWBAS
C C DIMENSION FILNAM(987)
C C CALL DBOPUT(NAMEDF,NEWBAS,FILNAM,RCODE)
C C DBOPEN=(RCODE.EQ.0)
99999 CONTINUE
      RETURN
      END
C C
C DB000010
C DB000020
C DB000030
C DB000040
C DB000050
C DB000060
C DB000070
C DB000080
C DB000090
C DB000100
C DB000110
C DB000120
C DB000130
C DB000140
C DB000150
C DB000160
C DB000170
C DB000180
C DB000190
C DB000200
C DB000210

```



```

C *****
C LOGICAL FUNCTION DBCLOS(NAMEDF,FILNAM,RCODE)
C +-----+
C DBCLOS -
C
C ROUTINE TO CALL DBCLUT TO CLOSE AN OPEN DATABASE
C
C RCODE AS RETURNED FROM DBCLUT
C +-----+
C
C INTEGER FILNAM,RCODE
C LOGICAL NAMEDF
C DIMENSION FILNAM(987)
C CALL DBCLUT(NAMEDF,FILNAM,RCODE)
C DBCLOS=(RCODE.EQ.O)
C RETURN
C END
C
DB000220
DB000230
DB000240
DB000250
DB000260
DB000270
DB000280
DB000290
DB000300
DB000310
DB000320
DB000330
DB000340
DB000350
DB000360
DB000370
DB000380
DB000390
DB000400
DB000410

```



```

C *****
C LOGICAL FUNCTION DBVINS(DNAME,TYPE,ARSIZE,RCODE)
C +-----+
C
C DBVINS -
C
C DBVINS INSERTS THE VARIABLE CONTAINED IN DNAME INTO THE
C OPEN DATABASE.
C
C RCODE IS A RETURN CODE;
C
C RCODE = 0 SUCCESSFUL INSERT
C RCODE = 1 NO OPEN DATABASE
C RCODE = 2 NODE ALREADY EXISTS
C RCODE = 3 NO MORE SPACE TO MAKE NODES
C RCODE = 4 WRONG TYPE
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER DNAME,TYPE,RCODE,DBGETN,BKTPTR,DBNSKR
C INTEGER RCODE,STYPE,ARSIZE
C DIMENSION DNAME(2)
C IF (DBCLSD) GO TO 11111
C IF (TYPE.GT.ARRTYP) GO TO 44444
C NODE=DBNSKR(DNAME,STYPE,BKTPTR,RCODE)
C IF (NODE.NE.O) GO TO 22222
C NODE=DBGETN(RCODE)
C IF (RCODE.NE.O) GO TO 33333
C
C +++ INSERT THE NODE IN THE DATABASE
C +++ LINK CHAINS UPWARD IN THE DATABASE TO PREVIOUS
C +++ NODES THAT HASHED TO THE SAME BUCKET. LINKF LINKSD800870
C +++ FORWARD TO PROVIDE A 2-WAY LINK BETWEEN THE NODES.DB000880

```



```

C...      *** THE NODE NUMBER IS INSERTED IN THE BUCKET AS A      DB000890
C...      *** POINTER FROM THE BUCKET TO THE NODE. IDENT(NODE,1)DB000900
C...      *** AND IDENT(NODE,2) CONTAIN THE DATUM NAME.  NODTYP DB000910
C...      *** IS SET TO THE NEGATIVE OF THE TYPE TO INDICATE NO DB000920
C...      *** DATA IS STORED YET.  IF NODTYP IS 3 THE EXPECTED DB000930
C...      *** ARRAY SIZE IS ENTERED IN DATUM.  DB000940
C...      DB000950
C...      LINK(NODE)=BUCKET(BKTPTR)      DB000960
C...      IF(LINK(NODE).NE.O) LINKF(LINK(NODE))=NODE      DB000970
C...      BUCKET(BKTPTR)=NODE      DB000980
C...      LINKF(NODE)=-BKTPTR      DB000990
C...      IDENT(NODE,1)=DNAME(1)      DB001000
C...      IDENT(NODE,2)=DNAME(2)      DB001010
C...      NODTYP(NODE)=-TYPE      DB001020
C...      DATUM(NODE)=O      DB001030
C...      IF (TYPE.EQ.ARRTYP) DATUM(NODE)=ARSIZE      DB001040
C...      CMTPTR(NODE)=O      DB001050
C...      GO TO 99999      DB001060
C...      DB001070
C...      DB001080
C...      DB001090
C...      DB001100
C...      DB001110
C...      DB001120
C...      DB001130
C...      DB001140
C...      DB001150
C...      DB001160
C...      DB001170
C...      DB001180
C...      DB001190
C...      DB001200
C...      DB001210
C...      DB001220
C...      DB001230
C...      DB001240
C...      DB001250
C...      DB001260

11111 CONTINUE
C...      NO OPEN DATA BASE
C...      RCODE=1
C...      GO TO 99999

22222 CONTINUE
C...      NODE ALREADY EXISTS
C...      RCODE=2
C...      GO TO 99999

33333 CONTINUE
C...      NO MORE SPACE TO MAKE NODES
C...      RCODE=3
C...      GO TO 99999

44444 CONTINUE
C...      TYPE HAS IMPROPER VALUE
C...      RCODE=4
C...      GO TO 99999

99999 CONTINUE
C...      DBVINS=(RCODE.EQ.O)
C...      RETURN
C...      END

```



```

C *****
C LOGICAL FUNCTION DBVDEL(DNAME,RCODE)
C +-----+
C DBVDEL -
C
C DELETES THE VARIABLE CONTAINED IN DNAME FROM THE OPEN
C DATABASE. IT ADJUSTS THE LINK AND LINKF AS APPROPRIATE
C
C RCODE IS THE RETURN CODE;
C
C RCODE = 1 NO OPEN DATA BASE
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER DNAME,DBNSKR,PRNODE,RCODE,STYPE
C DIMENSION DNAME(2)
C
C +++ SEARCH FOR THE NODE TO BE DELETED
C NODE=DBNSKR(DNAME,STYPE,PRNODE,RCODE)
C IF (DBCLSD) GO TO 11111
C DBVDEL=(NODE.NE.O)
C IF (.NOT.DBVDEL) GO TO 99999
C
C +++ MAKE IT UNAVAILABLE BY SETTING NODTYP = -9
C +++ IF THERE IS A LINK UPDATE THE LINK OF THE NODE TO
C +++ WHICH LINKED. COPY THE LINK INTO EITHER THE LINK OF
C +++ THE PREVIOUS NODE OR INTO THE BUCKET, DEPENDING ON
C +++ POSITION IN THE CHAIN. IF THE NODE IS AN ARRAY SET
C +++ ITS DATUM (ARRAY POINTER) TO THE NEGATIVE OF ITS
C +++ CURRENT VALUE. SIMILARLY FOR COMMENTS.
C
C IF (LINK(NODE).NE.O) LINKF(LINK(NODE))=LINKF(NODE)
C IF (RCODE.EQ.O) LINK(PRNODE)=LINK(NODE)

```



```

DB001740
DB001750
DB001760
DB001770
DB001780
DB001790
DB001800
DB001810
DB001820
DB001830
DB001840
DB001850
DB001860
DB001870
DB001880
DB001890

```

```

IF (RCODE.EQ.1) BUCKET(PRNODE)=LINK(NODE)
DATUM(NODE)=0
IF (NODTYP(NODE) .EQ. 3) DATUM(NODE)=-DATUM(NODE)
IF (CMTPTR(NODE) .NE. 0) CMTPTR(NODE)=-CMTPTR(NODE)
LINKF(NODE)=0
NODTYP(NODE)=-9
DELCNT=DELCNT+1
GO TO 99999

11111 CONTINUE
C...      +++ NO OPEN DATA BASE
          RCODE=1
          DBVDEL=.FALSE.
          GO TO 99999
99999 CONTINUE
          RETURN
          END

```



```

C *****
C LOGICAL FUNCTION INPUT(NAME, IVALUE, RCODE)
C +-----+
C INPUT -
C
C PUT AN INTEGER VALUE (IVALUE) IN THE DATABASE AT THE
C NODE ASSIGNED TO NAME.
C
C RCODE IS AS RETURNED FROM DBNDIN
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C 1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C 2 INTTYP, RELTYP, ARRTYP,
C 3 DBCLSD, DBACTV, FILEDB
C
C INTEGER
C 1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C 1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C 2 INTTYP, RELTYP, ARRTYP
C
C LOGICAL
C 3 DBCLSD, DBACTV, FILEDB
C
C DIMENSION
C 1 TITLE(16), BUCKET(32), IDENT(200, 2), NODTYP(200),
C DATUM(200), LINK(200), LINKF(200), CMTPTR(200)
C
C INTEGER NAME, IVALUE, RCODE
C LOGICAL DBNDIN
C DIMENSION NAME(2)
C INPUT=DBNDIN(NAME, INTTYP, IVALUE, RCODE)
99999 CONTINUE
      RETURN
      END

```



```

C *****
C LOGICAL FUNCTION IGET(NAME, IVALUE, RCODE)
C +-----+
C
C IGET -
C
C GET AN INTEGER VALUE FROM THE DATABASE AT THE NODE
C ASSIGNED TO NAME AND RETURN IT IN IVALUE.
C
C RCODE IS AS RETURNED FROM DBNDRD
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C 1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C 2 INTTYP, RELTYP, ARRTYP,
C 3 DBCLSD, DBACTV, FILEDB
C
C INTEGER
C 1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C 1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C 2 INTTYP, RELTYP, ARRTYP
C
C LOGICAL
C 3 DBCLSD, DBACTV, FILEDB
C
C DIMENSION
C 1 TITLE(16), BUCKET(32), IDENT(200, 2), NODTYP(200),
C DATUM(200), LINK(200), LINKF(200), CMTPTR(200)
C
C INTEGER NAME, IVALUE, RCODE
C LOGICAL DBNDRD
C DIMENSION NAME(2)
C IGET=DBNDRD(NAME, INTTYP, IVALUE, RCODE)
99999 CONTINUE
      RETURN
      END

```

DB002240
 DB002250
 DB002260
 DB002270
 DB002280
 DB002290
 DB002300
 DB002310
 DB002320
 DB002330
 DB002340
 DB002350
 DB002360
 DB002370
 DB002380
 DB002390
 DB002400
 DB002410
 DB002420
 DB002430
 DB002440
 DB002450
 DB002460
 DB002470
 DB002480
 DB002490
 DB002500
 DB002510
 DB002520
 DB002530
 DB002540
 DB002550
 DB002560
 DB002570


```

C *****
C LOGICAL FUNCTION RPUT(NAME,RVALUE,RCODE)
C +-----+
C
C RPUT -
C
C PUT A REAL VALUE (RVALUE) IN THE DATABASE AT THE NODE
C ASSIGNED TO NAME.
C
C RCODE IS AS RETURNED FROM DBNDIN
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER NAME,RCODE,ITEMP
C LOGICAL DBNDIN
C DIMENSION NAME(2)
C REAL RVALUE,RTEMP
C EQUIVALENCE (ITEMP,RTEMP)
C RTEMP = RVALUE
C RPUT=DBNDIN(NAME,RELTYP,ITEMP,RCODE)
C 99999 CONTINUE
C RETURN
C END

```



```

C *****
C LOGICAL FUNCTION RGET(NAME,RVALUE,RCODE)
C +-----+
C
C RGET-
C
C GET A REAL VALUE FROM THE DATABASE NODE ASSIGNED TO NAME
C AND RETURN IT IN RVALUE.
C
C RCODE IS AS RETURNED FROM DBNDRD
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER NAME,RCODE,ITEMP
C DIMENSION NAME(2)
C REAL RVALUE,RTEMP
C LOGICAL DBNDRD
C EQUIVALENCE (ITEMP,RTEMP)
C RGET=DBNDRD(NAME,RELTYP,ITEMP,RCODE)
C RVALUE=RTEMP
C 99999 CONTINUE
C RETURN
C END

```



```

C *****
C LOGICAL FUNCTION AGET(NAME,RARRAY,NGET,NSTORD,RCODE)
C +-----+
C
C AGET -
C
C GET THE VALUES OF AN ARRAY FROM THE ECS STORAGE AREA
C ASSIGNED TO NAME AND RETURN IN THE ARRAY RARRAY.
C
C RCODES 1-4 ARE AS RETURNED FROM DBNDRD
C RCODE = 5 NGET > NSTORD
C RCODE = 6 NGET < NSTORD
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER NAME,NGET,NSTORD,RCODE,POINTR,START
C DIMENSION NAME(2),RARRAY(987)
C
C LOGICAL DBNDRD,DBNARD
C DATA START /1/
C
C AGET=DBNDRD(NAME,ARRTYP,POINTR,RCODE)
C IF (RCODE.EQ.3) AGET=.FALSE.
C IF (RCODE.EQ.4) AGET=.FALSE.
C IF (RCODE.EQ.4) NSTORD=POINTR
C IF (.NOT.AGET) GO TO 99999
C AGET=DBNARD(POINTR,RARRAY,START,NGET,NSTORD)
C IF (NGET.GT.NSTORD) RCODE=5
C IF (NGET.LT.NSTORD) RCODE=6
C
C 99999 CONTINUE
C RETURN
C END

```



```

C *****
C LOGICAL FUNCTION APUT(NAME, RARRAY, NPUT, NSTORD, RCODE)
C +-----+
C
C      APUT =
C
C      PUT THE VALUES OF THE ARRAY (RARRAY) INTO THE ECS
C      STORAGE AREA ASSIGNED TO NAME.
C
C      RCODE = 0 SUCCESS
C      RCODE 1-3 AS IN DBNDIN
C      RCODE = 4 NPUT > NSTORD
C      RCODE = 5 NPUT < NSTORD
C
C +-----+
C
C      COMMON /DBCOM/
C
C      1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C      1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C      2 INTTYP, RELTYP, ARRTYP,
C      3 DBCLSD, DBACTV, FILEDB
C
C      INTEGER
C      1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C      1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C      2 INTTYP, RELTYP, ARRTYP
C
C      LOGICAL
C      3 DBCLSD, DBACTV, FILEDB
C
C      DIMENSION
C      1 DATUM(200), LINK(200), LINKF(200), CMTPTR(200),
C
C      INTEGER NAME, NPUT, NSTORD, RCODE, POINTR, START
C      LOGICAL DBNDRD, DBNAIN, DBNDIN, NEW
C      DIMENSION NAME(2), RARRAY(987)
C      DATA START /1/, NODAT /4/
C      APUT=DBNDRD(NAME, ARRTYP, POINTR, RCODE)
C      IF(RCODE.EQ.0) GO TO 1000
C      IF(RCODE.LT.4) GO TO 99999
C
C      1000 CONTINUE
C
C      NEW=(RCODE.EQ.NODAT)
C      APUT=DBNAIN(NEW, NAME, POINTR, RARRAY, START, NPUT, NSTORD)
C      IF (NEW.AND..APUT) APUT=DBNDIN(NAME, ARRTYP, POINTR, RCODE)
C      IF (APUT) GO TO 99999
C      IF (NPUT.GT.NSTORD) RCODE=4
C      IF (NPUT.LT.NSTORD) RCODE=5
C
C      99999 CONTINUE
C      RETURN
C      END

```



```

C *****
C LOGICAL FUNCTION IPUT(STRING)
C +-----+
C
C      TPUT -
C
C      PUT THE TITLE CONTAINED IN STRING INTO THE DATABASE.
C +-----+
C
C      *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C      *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C      COMMON /DEXCOM/
C      1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C      2 TERSE, ECHOMD, DXSC, CPWDXS, MDSC, CURMOD,
C      3 NULMSG, DXNCPW, NAMLEN,
C      4 CLRMOD, MUNPST, ERASE, UPDATE, UPAIN, GCNTRL
C
C      INTEGER
C      1 DXSC, CPWDXS, MDSC, CURMOD, NULMSG, DXNCPW,
C      2 NAMLEN, CLRMOD, MUNPST, ERASE, UPDATE, UPAIN
C
C      LOGICAL
C      1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C      2 TERSE, ECHOMD, GCNTRL
C
C      DIMENSION
C      COMMON /DBC/
C      1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C      1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C      2 INTTYP, RELTYP, ARRTYP,
C      3 DBCLSD, DBACTV, FILEDB
C
C      INTEGER
C      1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,
C      1 CMTPTR, NAVAIL, MAXNOD, MAXARR, NCHTIT, NCHCMT, DELCNT,
C      2 INTTYP, RELTYP, ARRTYP
C
C      LOGICAL
C      3 DBCLSD, DBACTV, FILEDB
C
C      DIMENSION
C      1 TITLE(16), BUCKET(32), IDENT(200,2), NODTYP(200),
C      DATUM(200), LINK(200), LINKF(200), CMTPTR(200)
C
C      INTEGER STRING
C      DIMENSION STRING(987)
C      IF (DBCLSD) GO TO 1111
C      LENTIT=NCHTIT/DXNCPW
C      DO 1000 I=1, LENTIT
C      TITLE(I)=STRING(I)
C      CONTINUE
C      TPUT=.TRUE.
C      GO TO 9999
C
C      1000

```



```
11111 CONTINUE  
      TPUT=.FALSE.  
      GO TO 99999  
99999 CONTINUE  
      RETURN  
      END
```

```
DB004700  
DB004710  
DB004720  
DB004730  
DB004740  
DB004750
```



```
1000      CONTINUE
          GO TO 9999
11111 CONTINUE
C...      +++ NO OPEN DATABASE
          TGET = .FALSE.
          GO TO 9999
99999 CONTINUE
          RETURN
          END
```

```
DB005230
DB005240
DB005250
DB005260
DB005270
DB005280
DB005290
DB005300
DB005310
```



```

C *****
C LOGICAL FUNCTION CMTGET(NAME,CMTSTR,RCODE)
C +-----+
C
C CMTGET -
C
C GET THE COMMENT STORED FOR VARIABLE NAME FROM THE
C ECS STORAGE AREA AND RETURN IT IN CMTSTR.
C
C RCODE IS A RETURN CODE;
C
C RCODE = 0 SUCCESS
C RCODE = 1 NO OPEN DATABASE
C RCODE = 2 NODE NOT IN DATABASE
C RCODE = 3 NO COMMENT STORED
C
C +-----+
C
C COMMON /DBCOM/
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C INTEGER CMTSTR,RCODE,DBNSKR,POINTR
C
C LOGICAL DBNCRD
C
C DIMENSION CMTSTR(987),NAME(2)
C
C IF (DBCLSD) GO TO 1111
C
C NODE=DBNSKR(NAME,IDUM1,IDUM2,RCODE)
C
C CMTGET=(NODE,NE.O)
C
C IF (.NOT.CMTGET) GO TO 2222
C
C POINTR=CMTPTR(NODE)
C
C CMTGET=DBNCRD(POINTR,CMTSTR)
C
C IF (.NOT.(CMTGET)) GO TO 3333
C
C RCODE=0
C GO TO 9999
C
C 1111 CONTINUE
C... +++ NO OPEN DATABASE
C RCODE=1

```


DB005790
DB005800
DB005810
DB005820
DB005830
DB005840
DB005850
DB005860
DB005870
DB005880
DB005890
DB005900

```
GO TO 99999
22222 CONTINUE    +++ NOT IN DATA BASE
C...             RCODE=2
                GO TO 99999
33333 CONTINUE    +++ NO COMMENT IS STORED
C...             RCODE=3
                GO TO 99999
99999 CONTINUE    RETURN
END
```


DB006380
DB006390
DB006400
DB006410
DB006420
DB006430
DB006440
DB006450
DB006460
DB006470
DB006480
DB006490
DB006500

```
RCODE=1
GO TO 99999
22222 CONTINUE
C...    +++ NOT IN DATA BASE
        RCODE=2
        GO TO 99999
33333 CONTINUE
C...    +++ NOT ABLE TO GET SPACE FOR COMMENT
        RCODE=3
        GO TO 99999
99999 CONTINUE
RETURN
END
```



```

C *****
C LOGICAL FUNCTION DBLOAD(RCODE)
C +-----+
C
C DBLOAD -
C
C LOAD AN EXISTING DATABASE FROM DISK.
C
C RCODE IS A RETURN CODE;
C
C RCODE = 0 DATABASE SUCCESSFULLY LOADED
C RCODE = 1 DATABASE FILE NOT FOUND
C RCODE = 2 FILE IS NOT A DEX DATABASE
C RCODE = 3 ECS STORAGE AREA IS FULL
C RCODE = 4 PREMATURE EOF ENCOUNTERED
C
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C++ THIS ROUTINE IS SITE DEPENDENT
C++ USES ECS
C +-----+
C ESTABLISH DATA BASE FOR 'DIRECT' ACCESS IN ECS
C +-----+
C
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1
C 2
C LOGICAL
C 1
C 2
C DIMENSION
C DXSC(1),CURMOD(2),NULMSG(1)
C
C *I/O DEVICES AND FILES
C
C COMMON /DEXFIL/
C 1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELPDV,
C 2 OUTDEV,INPDEV,NFWD,
C 3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
C 4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL
C
C MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELPDV,
C 1
C INTEGER
C 1

```



```

2      OUTDEV,INPDEV,
3      MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPFL,
4      OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL,NFWD5
C....  FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY.....
DIMENSION
1      MSGSFL(11),USEFIL(11),INFOFL(11),
3      DBSFIL(11),NEWSFL(11),HELPFL(11),
      OUTFIL(11),INPFIL(11),
2      LOADFL(11),NOT1FL(11),NOT2FL(11)
COMMON /DBCOM/
1      TITLE,CKET,IDENT,NODTYP,DATUM,LINK,LINKF,
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2      INITYP,RELTYP,ARRTYP,
3      DBCLSD,DBACTV,FILEDB
INTEGER
1      TITLE,CKET,IDENT,NODTYP,DATUM,LINK,LINKF,
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2      INITYP,RELTYP,ARRTYP
LOGICAL
3      DBCLSD,DBACTV,FILEDB
DIMENSION
1      TITLE(16),CKET(32),IDENT(200,2),NODTYP(200),
      DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
COMMON /DBDAIO/
1      MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
INTEGER
1      PRECS,ECSPTR,ECSLEN
DIMENSION
1      ECSPTR(2),ECSLEN(2)
INTEGER DBFCHK,CHKSTR,RCODE,WDSTIT,WDSCMT
LOGICAL DXCKDV,LOG,DBAPTR
DIMENSION ARRAY(200),DBFCHK(2),CHKSTR(2)
DATA     CHKSTR/4HDATA,4HBASE/,NAMSUB/4HDBLD/
DBLOAD = .TRUE.
IF (.NOT.DXCKDV(DBSDEV,DBSFIL)) GO TO 1111
REWIND DBSDEV
READ (DBSDEV,00001,END=44444) DBFCHK
IF (DBFCHK(1).NE.CHKSTR(1)) GO TO 22222
IF (DBFCHK(2).NE.CHKSTR(2)) GO TO 22222
CALL DBINIT
WDSTIT=NCHTIT/DXNCPW
WDSCMT=NCHCMT/DXNCPW
C....  +++ LOAD THE DATABASE FROM THE DBSDEV
      READ (DBSDEV,00002,END=44444) (TITLE(I),I=1,WDSTIT)
      READ (DBSDEV,00003,END=44444) NAVAIL,DELCNT,MAXECS,NEXECS,
      PRECS,CKET
+
      DO 1000 I=1,MAXNOD
      READ (DBSDEV,00004,END=44444) IDENT(I,1),IDENT(I,2),
      NODTYP(I),DATUM(I),LINK(I),LINKF(I),CMTPTR(I)
+

```



```

1000      CONTINUE
      DO 4000 I=1,MAXNOD
      *** LOAD THE ARRAYS AND COMMENTS
      *** LOAD ARRAYS
      IF (NODTYP(1).NE. 3) GO TO 2000
      READ (DBSDEV,00005,END=4444) L,(ARRAY(J),J=1,L)
      IF (.NOT.(DBAPTR(IPPOINT,IDUM,1))) GO TO 33333
      DATUM(I)=IPPOINT
      CALL WRITEC(L,IPPOINT,1,ECSPTR(1),ECSLEN(1))
      IF(.NOT.DBAPTR(IPPOINT,IDUM,L)) GO TO 33333
      CALL WRITEC(ARRAY,IPPOINT,L,ECSPTR(1),ECSLEN(1))
      CONTINUE
      *** LOAD THE COMMENTS
      IF(CMTPTR(I).EQ.O) GO TO 3000
      READ (DBSDEV,00006,END=4444) (ARRAY(J),
      J=1,WDESCMT)
      IF(.NOT.DBAPTR(IPPOINT,IDUM,WDESCMT)) GO TO 33333
      CALL WRITEC(ARRAY,IPPOINT,16,ECSPTR(1),ECSLEN(1))
      CMTPTR(I)=IPPOINT
      CONTINUE
      CONTINUE
      RCODE=0
      GO TO 99999
      +
      3000      CONTINUE
      4000      CONTINUE
      11111 CONTINUE
      C...      CALL DXMSGZ(NAMSUB,2)
      RCODE=1
      GO TO 99999
      22222 CONTINUE
      C...      *** FILE IS NOT A DEX DATABASE
      RCODE=2
      CALL DXMSGZ(NAMSUB,3)
      GO TO 99999
      33333 CONTINUE
      C...      *** DIRECT ACCESS STORAGE FULL
      RCODE=3
      GO TO 99999
      44444 CONTINUE
      C...      *** PREMATURE EOF ENCOUNTERED
      RCODE=4
      GO TO 99999
      99999 CONTINUE
      DBLOAD=(RCODE.EQ.O)
      RETURN
      00001 FORMAT (1X,2A4)
      00002 FORMAT (1X,16A4)

```


DBLO1420
DBLO1430
DBLO1440
DBLO1450
DBLO1460
DBLO1470

00003 FORMAT (1X,5I10/(16I5))
C++ THE NEXT TWO FORMATS ARE SITE DEPENDENT
00004 FORMAT (2X,2A4,1X,I4,1X,Z8,1X,4I5)
00005 FORMAT (2X,Z8,1X,Z8,1X,Z8)
00006 FORMAT (2X,16A4)
END


```

C *****
C LOGICAL FUNCTION DBSAVE(FILNAM)
C +-----+
C
C DBSAVE -
C
C SAVE A DATABASE FOR FUTURE USE.
C
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C+++ THIS ROUTINE IS SITE DEPENDENT
C+++ FILE MANIPULATION
C +-----+
C *THE COMMON CONTROL PARAMETERS, CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C 2 TERSE, ECHOMD, DXSC, CPWDXS, MDSC, CURMOD,
C 3 NULMSG, DXNCPW, NAMLEN,
C 4 CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE, GCNTRL
C
C INTEGER
C 1 DXSC, CPWDXS, MDSC, CURMOD, NULMSG, DXNCPW,
C 2 NAMLEN, CLRMOD, MUNPST, ERASE, UPDATE, UPAINTE
C
C LOGICAL
C 1 DXMODE, LOADED, MDPEND, DXREQS, DLREQS, KEYBRD,
C 2 TERSE, ECHOMD, GCNTRL
C
C DIMENSION
C DXSC(1), CURMOD(2), NULMSG(1)
C
C *I/O DEVICES AND FILES
C
C COMMON /DEXFIL/
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDV, HELPDV,
C 2 OUTDEV, INPDEV, NFWD,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL
C
C INTEGER
C 1 MSGSDV, USEDEV, INFODV, DBSDEV, NEWSDV, HELPDV,
C 2 OUTDEV, INPDEV,
C 3 MSGSFL, USEFIL, INFOFL, DBSFIL, NEWSFL, HELPFL,
C 4 OUTFIL, INPFIL, LOADFL, NOT1FL, NOT2FL, NFWD
C
C FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY.....
C DIMENSION
C 1 MSGSFL(11), USEFIL(11), INFOFL(11),
C DBSFIL(11), NEWSFL(11), HELPFL(11),
C 3 OUTFIL(11), INPFIL(11),
C 2 LOADFL(11), NOT1FL(11), NOT2FL(11)
C
C COMMON /DBCOM/
C 1 TITLE, BUCKET, IDENT, NODTYP, DATUM, LINK, LINKF,

```



```

1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT, DBLO1950
2      INTTYP,RELTYP,ARRTYP, DBLO1960
3      DBCLSD,DBACTV,FILEDB DBLO1970
      INTEGER DBLO1980
1      TITLE ,BUCKET,IDENT ,NODTYP,OATUM ,LINK ,LINKF , DBLO1990
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT, DBLO2000
2      INTTYP,RELTYP,ARRTYP DBLO2010
      LOGICAL DBLO2020
3      DBCLSD,DBACTV,FILEDB DBLO2030
      DIMENSION TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200), DBLO2040
1      DATUM(200),LINK(200),LINKF(200),CMTPTR(200) DBLO2050
      COMMON /DBDATA/ DBLO2060
1      MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN DBLO2070
      INTEGER PRECS,ECSPTR,ECSLEN DBLO2080
1      DIMENSION DBLO2090
      ECSPTR(2),ECSLEN(2) DBLO2100
1      INTEGER RCODE,WDSTIT,WDSCMT,FILNAM,DBFCHK DBLO2110
      LOGICAL DBWARD,CMTGET,LOG DBLO2120
      LOGICAL DXCKDV DBLO2130
      DIMENSION FILNAM(987),NAME(2),ITEMP(200),TEMP(200),DBFCHK(2) DBLO2140
      EQUIVALENCE (ITEMP(1),TEMP(1)) DBLO2150
      DATA NAMSUB /4HSAVE/ DBLO2160
      DATA DBFCHK /4HDATA,4HBASE/ DBLO2170
      DBLO2180
      DBLO2190
      DBLO2200
      DBLO2210
      DBLO2220
      DBLO2230
      DBLO2240
      DBLO2250
      DBLO2260
      DBLO2270
      DBLO2280
      DBLO2290
      DBLO2300
      DBLO2310
      DBLO2320
      DBLO2330
      DBLO2340
      DBLO2350
      DBLO2360
      DBLO2370
      DBLO2380
      DBLO2390
      DBLO2400
      DBLO2410

```

C

C...

1000
C...

2000
C...


```

      IF (CMTPTR(I).EQ.O) GO TO 3000
      NAME(1)=IDENT(I,1)
      NAME(2)=IDENT(I,2)
      LOG=CMTGET(NAME,ITEMP,RCODE)
      WRITE (DBSDEV,00006) (ITEMP(J),J=1,WDSGMT)

3000  CONTINUE
4000  CONTINUE
      REWIND DBSDEV
      C...      *** ANNOUNCE THAT DATABASE HAS BEEN SAVED
                CALL DXMSGZ(NAMSUB,1)
                GO TO 9999
11111 CONTINUE
      C...      *** FAILED TO SAVE DATABASE
                DBSAVE=.FALSE.
                CALL DXMSGI(NAMSUB,2,IRC)
                GO TO 9999
99999 CONTINUE
      RETURN
00001 FORMAT (1H ,2A4)
00002 FORMAT (1H ,16A4)
00003 FORMAT (1H ,5I10/(16I5))
      C...      *** THE NEXT TWO FORMATS ARE MACHINE DEPENDENT
00004 FORMAT (2H *,2A4,1H*,I4,1H*,Z8,1H*,4I5)
00005 FORMAT (2H *,Z8,1H*,Z8,1H*,Z8,1H*,Z8,1H*)
00006 FORMAT (2H *,16A4,1H*)
      END

```

DBL02420
 DBL02430
 DBL02440
 DBL02450
 DBL02460
 DBL02470
 DBL02480
 DBL02490
 DBL02500
 DBL02510
 DBL02520
 DBL02530
 DBL02540
 DBL02550
 DBL02560
 DBL02570
 DBL02580
 DBL02590
 DBL02600
 DBL02610
 DBL02620
 DBL02630
 DBL02640
 DBL02650
 DBL02660
 DBL02670


```

C
C *****
C INTEGER FUNCTION DBHASH(DNAME)
C +-----+
C
C DBHASH -
C
C HASH DNAME TO A NUMBER BETWEEN 1 AND 32 TO BE USED AS
C AN INDEX TO ITS LOCATION IN THE DATABASE.
C
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C+++
C THIS ROUTINE USES THE INTERNAL CODE VALUES FOR EACH
C... CHARACTER TO SUM THEM AND TRUNCATE THE SUM TO OBTAIN
C... VALUES IN THE RANGE OF 1 TO 32.
C... THE SUMMATION CEASES WHEN THE FIRST BLANK IN THE NAME
C... IS ENCOUNTERED.
C +-----+
C
C INTEGER DNAME,NAMWRD,IFWORD,ZBLANK
C LOGICAL*1 NAMBYT,IFBYTE
C DIMENSION DNAME(2),NAMWRD(2)
C DIMENSION NAMBYT(8),IFBYTE(4)
C EQUIVALENCE (NAMWRD(1),NAMBYT(1)),(IFWORD,IFBYTE(1))
C DATA ZBLANK /ZOOOOOOO40/
C NAMWRD(1)=DNAME(1)
C NAMWRD(2)=DNAME(2)
C IFWORD=ZBLANK
C DBHASH=0
C DO 1000 I=1,8
C   IFBYTE(4)=NAMBYT(I)
C   IF (IFWORD.EQ.ZBLANK) GO TO 2000
C   DBHASH=DBHASH+IFWORD
C 1000 CONTINUE
C 2000 CONTINUE
C   DBHASH=IABS(MOD(DBHASH,32))+1
C 99999 CONTINUE
C   RETURN
C END
C
DBL02680
DBL02690
DBL02700
DBL02710
DBL02720
DBL02730
DBL02740
DBL02750
DBL02760
DBL02770
DBL02780
DBL02790
DBL02800
DBL02810
DBL02820
DBL02830
DBL02840
DBL02850
DBL02860
DBL02870
DBL02880
DBL02890
DBL02900
DBL02910
DBL02920
DBL02930
DBL02940
DBL02950
DBL02960
DBL02970
DBL02980
DBL02990
DBL03000
DBL03010
DBL03020
DBL03030
DBL03040
DBL03050
DBL03060
DBL03070

```



```

C *****
C LOGICAL FUNCTION DBAPTR(POINTR,PREPTR,NEED)
C +-----+
C
C DBAPTR -
C
C DATABASE ARRAY POINTER - OBTAINS THE POINTER TO THE
C NEXT AVAILABLE LOCATION IN THE ECS AREA IN POINTR.
C
C PREPTR IS A POINTER TO THE LOCATION OF THE LAST ARRAY
C OR COMMENT ENTERED IN THE STORAGE AREA.
C
C NEED IS THE NUMBER OF LOCATIONS WHICH NEED TO BE ENTERED.
C +-----+
C
C +++ THIS ROUTINE IS MACHINE DEPENDENT
C +++ THIS ROUTINE IS SITE DEPENDENT
C +++ USES ECS
C +-----+
C
COMMON /DBCOM/
1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2 INTTYP,RELTYP,ARRTYP,
3 DBCLSD,DBACTV,FILEDB
C
INTEGER
1 TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2 INTTYP,RELTYP,ARRTYP
C
LOGICAL
3 DBCLSD,DBACTV,FILEDB
C
DIMENSION
1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
COMMON /DBDAIO/
1 MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
C
INTEGER
1 PRECS,ECSPTR,ECSLEN
C
DIMENSION
1 ECSPTR(2),ECSLEN(2)
C
INTEGER POINTR,PREPTR
DATA
1 NAMSUB/4HAPTR/
C...
1 DBAPTR=((NEXECS+NEED-1).LE.MAXECS)
1 IF (.NOT.DBAPTR) GO TO 8888
1 +++ SET POINTER TO NEXT AVAILABLE ECS, AND PREPTR TO THE
C...

```



```

C...      *** PREVIOUS ECS LOCATION.
          POINTR=NEXECS
          PREPTR=PRECS
          *** UPDATE THE NEXECS AND PRECS FOR NEXT TIME.
          PRECS=NEXECS
          NEXECS=NEXECS+NEED
          GO TO 99999
88888 CONTINUE
          CALL DXMSGZ(NAMSUB,1)
          DBAPTR=.FALSE.
          GO TO 99999
99999 CONTINUE
          RETURN
          END
          DBL03550
          DBL03560
          DBL03570
          DBL03580
          DBL03590
          DBL03600
          DBL03610
          DBL03620
          DBL03630
          DBL03640
          DBL03650
          DBL03660
          DBL03670
          DBL03680

```



```

C *****
C LOGICAL FUNCTION DBCPTR(POINTR,PREPTR,NEED)
C +-----+
C
C DBCPTR -
C
C DATABASE COMMENT POINTER - OBTAINS THE POINTER TO THE
C NEXT AVAILABLE LOCATION IN THE ECS AREA IN POINTR.
C
C PREPTR IS A POINTER TO THE LOCATION OF THE LAST ARRAY
C OR COMMENT ENTERED IN THE STORAGE AREA.
C
C NEED IS THE NUMBER OF LOCATIONS WHICH NEED TO BE ENTERED.
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C++ THIS ROUTINE IS SITE DEPENDENT
C++ USES ECS
C++ INTEGER POINTR,PREPTR
C++ LOGICAL DBAPTR
C++ DBCPTR=DBAPTR(POINTR,PREPTR,NEED)
99999 CONTINUE
      RETURN
      END

```

```

DBL03690
DBL03700
DBL03710
DBL03720
DBL03730
DBL03740
DBL03750
DBL03760
DBL03770
DBL03780
DBL03790
DBL03800
DBL03810
DBL03820
DBL03830
DBL03840
DBL03850
DBL03860
DBL03870
DBL03880
DBL03890
DBL03900
DBL03910
DBL03920
DBL03930
DBL03940

```



```

C *****
C LOGICAL FUNCTION DBNAIN(NEW,NAME,POINTR,RARRAY,START,NPUT,
1 NSTORD)
C +-----+
C DBNAIN -
C
C DATABASE NODE ARRAY IN - STORES THE VALUES OF THE ARRAY
C RARRAY INTO THE ECS STORAGE AREA FOR VARIABLE NAME.
C
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C THIS ROUTINE IS SITE DEPENDENT
C USES ECS
C +-----+
C
COMMON /DBDAID/
1 MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
1 INTEGER
1 PRECS,ECSPTR,ECSLEN
1 DIMENSION
1 ECSPTR(2),ECSLEN(2)
1 INTEGER POINTR,START,NPUT,NSTORD,PREPTR,DBNSKR
1 LOGICAL NEW,DBAPTR
1 DIMENSION RARRAY(NPUT),NAME(2)
1 IF (NEW) GO TO 1000
1 +++ ARRAY IS ALREADY STORED USE SAME AREA.
1 CALL READC(NSTORD,POINTR,1,ECSPTR(1),ECSLEN(1))
1 DBNAIN=(NPUT,EQ,NSTORD)
1 IF (.NOT.(DBNAIN)) GO TO 99999
1 CALL WRITEC(RARRAY,POINTR+3,MINO(NPUT,NSTORD),ECSPTR(1),
1 ECSLEN(1))
1 GO TO 99999
1000 CONTINUE
NSTORD=NPUT
C... +++ GET SPACE FOR ARRAY ITS LENGTH AND POINTER INFORMATION
DBNAIN=DBAPTR(POINTR,PREPTR,NSTORD+3)
IF (.NOT.(DBNAIN)) GO TO 99999
C... +++ PUT IN THE NUMBER STORED
CALL WRITEC(NSTORD,POINTR,1,ECSPTR(1),ECSLEN(1))
C... +++ PUT IN A POINTER TO THE PREVIOUS ARRAY OR COMMENT STORED
CALL WRITEC(PREPTR,POINTR+1,1,ECSPTR(1),ECSLEN(1))
C... +++ PUT IN A POINTER BACK TO THE NODE
NODE=DBNSKR(NAME,IDUM1,IDUM2,IDUM3)
CALL WRITEC(NODE,POINTR+2,1,ECSPTR(1),ECSLEN(1))

```



```

C...      1      ++ PUT IN THE ARRAY VALUES
            CALL WRITEC(RARRAY,POINTR+3,MINO(NPUT,NSTORD),
            1      ECSPTR(1),ECSLEN(1))
            IF (NPUT.EQ.NSTORD) GO TO 99999
            IBEG=NPUT+1
            DO 2000 I=IBEG,NSTORD
            CALL WRITEC(O.O,POINTR+3+I,1,ECSPTR(1),ECSLEN(1))
            CONTINUE
            GO TO 99999
2000      CONTINUE
99999      CONTINUE
            RETURN
            END
DBLO4420
DBLO4430
DBLO4440
DBLO4450
DBLO4460
DBLO4470
DBLO4480
DBLO4490
DBLO4500
DBLO4510
DBLO4520
DBLO4530

```



```

C *****
C LOGICAL FUNCTION DBNARD(POINTR, RARRAY, START, NGET, NSTORD)
C +-----+
C DBNARD -
C
C DATABASE NODE ARRAY READ - READS THE CONTENTS OF THE
C ARRAY POINTED TO BY POINTR FROM THE ECS STORAGE AREA
C INTO RARRAY.
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C+++ THIS ROUTINE IS SITE DEPENDENT
C+++ USES ECS
C+++ COMMON /DBDAIO/
C 1 MAXECS, NEXECS, PRECS, ECSPTR, ECSLEN
C INTEGER
C 1 PRECS, ECSPTR, ECSLEN
C DIMENSION
C 1 ECSPTR(2), ECSLEN(2)
C INTEGER POINTR, START, NGET, NSTORD
C DIMENSION RARRAY(987)
C... ++ READ THE NUMBER STORED
C... CALL READEC(NSTORD, POINTR, 1, ECSPTR(1), ECSLEN(1))
C... ++ READ THE ARRAY ELEMENTS
C... CALL READEC(RARRAY, POINTR+3, MINO(NSTORD, NGET), ECSPTR(1), ECSLEN(1))
C IF (NSTORD .GE. NGET) GO TO 2000
C IB=NSTORD+1
C DO 1000 I=IB, NGET
C RARRAY(I)=0.
C CONTINUE
1000 CONTINUE
2000 CONTINUE
C DBNARD=.TRUE.
C GO TO 99999
99999 CONTINUE
C RETURN
C END

```



```

C
C *****
C LOGICAL FUNCTION DBNCRD(POINTR,CMTSTR)
C +-----+
C
C DBNCRD -
C
C DATABASE NODE COMMENT READ - READS THE COMMENT POINTED
C TO BY POINTR FROM THE ECS STORAGE AREA INTO CMTSTR.
C
C +-----+
C
C+++ THIS ROUTINE IS MACHINE DEPENDENT
C+++ THIS ROUTINE IS SITE DEPENDENT
C+++ USES ECS
C +-----+
C
C *****
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C COMMON /DBCOM/
C 1 TITLE,BUCKET,IDENT,NODTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE,BUCKET,IDENT,NODTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
C
C COMMON /DBDAIO/
C 1 MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
C
C INTEGER

```



```

1      DIMENSION      PRECS,ECSPTR,ECSLEN
1      INTEGER      POINTR,CMTSTR,CMTWDS
1      DIMENSION CMTSTR(987)
1      CMTWDS=NCHCMT/DXNCPW
1      DBNCRD=(POINTR.GT.O)
1      C...      *** READ THE COMMENT
1      IF (DBNCRD) CALL READEC(CMTSTR,POINTR+2,CMTWDS,ECSPTR(1),
1      ECSLEN(1))
1      99999 CONTINUE
1      RETURN
1      END

```

```

DBL05400
DBL05410
DBL05420
DBL05430
DBL05440
DBL05450
DBL05460
DBL05470
DBL05480
DBL05490
DBL05500
DBL05510
DBL05520

```



```

C *****
C LOGICAL FUNCTION DBNCIN(POINTR,CMTSTR,NODE)
C +-----+
C DBNCIN -
C
C DATABASE NODE COMMENT IN - ENTERS THE COMMENT CONTAINED
C IN CMTSTR INTO THE ECS STORAGE AREA AND RETURNS THE
C POINTER TO ITS LOCATION IN POINTR. THE COMMENT IS
C LINKED BACK TO ITS NODE.
C +-----+
C
C THIS ROUTINE IS MACHINE DEPENDENT
C+++ THIS ROUTINE IS SITE DEPENDENT
C+++ USES ECS
C +-----+
C
C *THE COMMON CONTROL PARAMETERS,CHARACTERS, AND
C *GRAPHIC DISPLAY CONTROL PARAMETERS
C
C COMMON /DEXCOM/
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,DXSC,CPWDXS,MDSC,CURMOD,
C 3 NULMSG,DXNCPW,NAMLEN,
C 4 CLRMOD,MUNPST,ERASE,UPDATE,UPAINT,GCNTRL
C
C INTEGER
C 1 DXSC,CPWDXS,MDSC,CURMOD,NULMSG,DXNCPW,
C 2 NAMLEN,CLRMOD,MUNPST,ERASE,UPDATE,UPAINT
C
C LOGICAL
C 1 DXMODE,LOADED,MDPEND,DXREQS,DLREQS,KEYBRD,
C 2 TERSE,ECHOMD,GCNTRL
C
C DIMENSION
C COMMON /DBCOM/
C 1 TITLE,BUCKET,IDENT,NOOTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP,
C 3 DBCLSD,DBACTV,FILEDB
C
C INTEGER
C 1 TITLE,BUCKET,IDENT,NOOTYP,DATUM,LINK,LINKF,
C 1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
C 2 INTTYP,RELTYP,ARRTYP
C
C LOGICAL
C 3 DBCLSD,DBACTV,FILEDB
C
C DIMENSION
C 1 DATUM(200),LINK(200),LINKF(200),CMTPTR(200),
C COMMON /DBDAIO/

```



```

1      INTEGER      MAXECS, NEXECS, PRECS, ECSPTR, ECSLEN
1      DIMENSION    PRECS, ECSPTR, ECSLEN
1      INTEGER      ECSPTR(2), ECSLEN(2)
      LOGICAL      DBCPTR
      DIMENSION    CMTSTR(987)
      CMTWDS=NCHCMT/DXNCPW
      DBNCIN=(POINTR.GT.O)
      C...          *** IF ALREADY HAVE A POINTER DON'T GET ANOTHER
      C...          *** OTHERWISE GET THE POINTER TO THE NEXT AVAILABLE ECS
      IF (.NOT.(DBNCIN)) DBNCIN=DBCPTR(POINTR,PREPTR,CMTWDS+2)
      IF (.NOT.(DBNCIN)) GO TO 99999
      C...          *** PUT IN A POINTER BACK TO PREVIOUS COMMENT OR ARRAY
      C...          *** LOCATION IN THE ECS AREA. PUT IN A POINTER BACK TO
      C...          *** THE NODE, AND THEN PUT IN THE COMMENT.
      CALL WRITEC(PREPTR,POINTR,1,ECSPTR(1),ECSLEN(1))
      CALL WRITEC(NODE,POINTR+1,1,ECSPTR(1),ECSLEN(1))
      CALL WRITEC(CMTSTR,POINTR+2,CMTWDS,ECSPTR(1),ECSLEN(1))
99999  CONTINUE
      RETURN
      END

```

```

DBL06000
DBL06010
DBL06020
DBL06030
DBL06040
DBL06050
DBL06060
DBL06070
DBL06080
DBL06090
DBL06100
DBL06110
DBL06120
DBL06130
DBL06140
DBL06150
DBL06160
DBL06170
DBL06180
DBL06190
DBL06200
DBL06210
DBL06220

```



```

C*****
C LOGICAL FUNCTION DBXEC(S>IDUM)
C +-----+
C DBXEC =
C
C DATABASE EXPAND ECS - EXPANDS THE DATABASE SIMULATED ECS
C STORAGE AREA BY INCREMENTS OF 2000.
C
C +-----+
C
C COMMON /DBDAIO/
C 1 MAXECS,NEXECS,PRECS,ECSPTR,ECSLEN
C INTEGER
C 1 PRECS,ECSPTR,ECSLEN
C 1 DIMENSION
C 1 ECSPTR(2),ECSLEN(2)
C LOGICAL GETMN,FREEMN,LOG
C INTEGER RFODE
C COMMON /CODE/ RFODE
C
C +++ SAVE THE OLD MAXECS AND INCREMENT MAXECS BY 2000
C
C ECSLEN(1)=MAXECS
C MAXECS=MAXECS+2000
C ECSLEN(2)=MAXECS
C
C +++ GET STORAGE VIA GETMAIN AND COPY THE DATABASE ARRAYS
C
C DBXEC=GETMN(MAXECS,ECSPTR(2))
C IF (.NOT. DBXEC) GO TO 1111
C KLEN = ECSLEN (1)
C DO 1000 I=1,KLEN
C CALL READC(RARRAY,I,1,ECSPTR(1),ECSLEN(1))
C CALL WRITEC(RARRAY,I,1,ECSPTR(2),ECSLEN(2))
C 1000 CONTINUE
C
C +++ FREE THE ORIGINAL STORAGE AREA VIA FREEMAIN
C
C LOG=FREEMN(ECSLEN(1),ECSPTR(1),RFODE)
C ECSPTR(1)=ECSPTR(2)
C ECSLEN(1)=ECSLEN(2)
C GO TO 9999
C
C 1111 MAXECS = MAXECS - 2000
C 9999 RETURN
C END

```



```

C *****DXA00010
SUBROUTINE DXABND-----+
C
C
C      DXABND -
C
C      DEX ABNORMAL END - WHEN AN ABNORMAL END OCCURS THE
C      ABEND CONDITION IS TRAPPED BY THE ASSEMBLY LANGUAGE
C      ROUTINE ABTRAP, AND CONTROL IS PASSED TO THIS ROUTINE
C      UPON ENTRY THE VALUE OF RCODE INDICATES THE ROUTINE
C      WHICH CAUSED THE ABNORMAL END.
C
C      THE CONDITION IS ANNOUNCED TO THE USER.
C
C      THE USER IS GIVEN THE OPTION OF SAVING ANY OPEN
C      DATABASE.
C
C      THE USER IS THEN GIVEN THE OPTION OF CONTINUING DEX
C      OPERATION, OR STOPPING. THE LATTER IS RECOMMENDED.
C
C-----+
C+++ THIS ROUTINE IS MACHINE DEPENDENT
C+++ THIS ROUTINE IS SITE DEPENDENT
C-----+
C
C      *I/O DEVICES AND FILES
C...
COMMON /DEXFIL/
1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELDPDV,
2 OUTDEV,INPDEV,NFWD,
3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL
INTEGER
1 MSGSDV,USEDEV,INFODV,DBSDEV,NEWSDV,HELDPDV,
2 OUTDEV,INPDEV,
3 MSGSFL,USEFIL,INFOFL,DBSFIL,NEWSFL,HELPL,
4 OUTFIL,INPFIL,LOADFL,NOT1FL,NOT2FL,NFWD
C... FILE NAME DIMENSIONING ADJUSTED FOR CDC AND CU ONLY....
DIMENSION
1 MSGSFL(11),USEFIL(11),INFOFL(11),HELPL(11),
3 DBSFIL(11),NEWSFL(11),HELPL(11),
2 OUTFIL(11),INPFIL(11),LOADFL(11),NOT1FL(11),NOT2FL(11)
C... DATABASE COMMON
COMMON /DBCOM/
1 TITLE,BUCKET,IDENT,NOOTYP,DATUM,LINK,LINKF,
1 CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2 INTTYP,RELTYP,ARRTYP,

```



```

3      DBCLSD,DBACTV,FILEDB
      INTEGER
1      TITLE ,BUCKET,IDENT ,NODTYP,DATUM ,LINK ,LINKF ,
1      CMTPTR,NAVAIL,MAXNOD,MAXARR,NCHTIT,NCHCMT,DELCNT,
2      INTTYP,RELTYP,ARRTYP
      LOGICAL
3      DBCLSD,DBACTV,FILEDB
      DIMENSION
1      TITLE(16),BUCKET(32),IDENT(200,2),NODTYP(200),
      DATUM(200),LINK(200),LINKF(200),CMTPTR(200)
      COMMON /CODE/ RFCODE
      LOGICAL SAVEFL,DBSAVE
      INTEGER FILENM
      DIMENSION FILENM(5)
      DATA FILENM/4H ,4H ,4H DAT,4HABAS,4HE A /
      DATA NAMSUB/4HABND/
C      ANNOUNCE THAT AN ABNORMAL TERMINATION HAS OCCURED.
C... IF RFCODE = 1 THEN ANNOUNCE THAT THE ERROR WAS IN ROUTINE LOAD
C... WHILE LOADING A MODULE.
C... IF RFCODE = 2 THEN ANNOUNCE THAT THE ERROR WAS IN ROUTINE START
C... WHILE STARTING A MODULE.
C... IF RFCODE = 3 THEN ANNOUNCE THAT THE ERROR WAS IN ROUTINE FREEMN.
C... IF RFCODE = 4 THEN ANNOUNCE THAT THE ERROR WAS IN THE MODULE.
C
      WRITE (OUTDEV,00001)
      IF (RFCODE .EQ. 1) WRITE (OUTDEV,00002)
      IF (RFCODE .EQ. 2) WRITE (OUTDEV,00003)
      IF (RFCODE .EQ. 3) WRITE (OUTDEV,00004)
      IF (RFCODE .EQ. 4) WRITE (OUTDEV,00005)
      IF (.NOT.DBCLSD) GO TO 7777
C
      IF THERE WAS AN OPEN DATABASE SAVE IT
C
      WRITE (OUTDEV,00006)
      READ (INPDEV,00007) IANS
      IF (IANS .EQ. 0) GO TO 88888
      WRITE (OUTDEV,00008)
      READ (INPDEV,00009) (FILENM(I),I=1,2)
      SAVEFL=DBSAVE(FILENM)
      IF (SAVEFL) WRITE (OUTDEV,00010)
      IF (.NOT. SAVEFL) WRITE (OUTDEV,00011)
      GO TO 88888
C
      ANNOUNCE THAT THERE IS NO OPEN DATABASE
C
7777 CONTINUE
      WRITE (OUTDEV,00012)

```



```

C          CHECK IF THE USER WANTS TO RETURN TO DEX OR STOP
C
C      88888 CONTINUE
C          WRITE (OUTDEV,00013)
C          READ (INPDEV,00007) IANS
C          IF (IANS.EQ.0) GO TO 99999
C              CALL DXMAJJC
C              STOP
C      99998 RETURN
C
C          THATS IT THE USER CHOSE TO STOP
C
C      99999 CONTINUE
C          STOP
C
C          FORMAT STATEMENTS
C
C      00001 FORMAT (1X,'AN ABNORMAL TERMINATION OF EXECUTION HAS OCCURED.',/,
C      1      'THE SYSTEM ABEND HAS BEEN INTERCEPTED BY DEX.',/)
C      00002 FORMAT (1X,'THE ABEND WAS CAUSED IN ROUTINE LOAD WHILE LOADING',/
C      1      'A MODULE PROGRAM.',/)
C      00003 FORMAT (1X,'THE ABEND WAS CAUSED IN ROUTINE START WHILE STARTING',
C      1      'A MODULE PROGRAM.',/)
C      00004 FORMAT (1X,'THE ABEND WAS CAUSED IN ROUTINE FREEMN',/)
C      00005 FORMAT (1X,'THE ABEND WAS CAUSED BY THE EXECUTING MODULE',/)
C      00006 FORMAT (1X,'DEX IS NOW UNPREDICTABLE. DO YOU WANT TO SAVE YOUR',/
C      1      'OPEN DATABASE. (ENTER 1=YES OR 0=NO)',/)
C      00007 FORMAT (I1)
C      00008 FORMAT (1X,'ENTER AN 8 CHARACTER FILENAME FOR YOUR DATABASE.',/,
C      1      'A FILETYPE OF DATABASE IS ASSUMED.',/)
C      00009 FORMAT (244)
C      00010 FORMAT (1X,'YOUR DATABASE HAS BEEN SAVED.',/)
C      00011 FORMAT (1X,'UNABLE TO SAVE YOUR DATABASE.',/)
C      00012 FORMAT (1X,'THERE IS NO OPEN DATABASE.',/)
C      00013 FORMAT (1X,'SINCE THE SYSTEM AND POSSIBLY DEX HAVE BEEN HAMMERED',
C      1      'FURTHER EXECUTION WILL HAVE UNPREDICTABLE RESULTS.',/,
C      2      'WE RECOMMEND THAT YOU STOP AND RESTART DEX. DO YOU',/
C      3      'WISH TO CONTINUE WITH DEX OR STOP?',/,
C      4      ' (ENTER 1 TO CONTINUE, OR 0 TO STOP)',/)
C          END
C      00014
C      00015
C      00016
C      00017
C      00018
C      00019
C      00020
C      00021
C      00022
C      00023
C      00024
C      00025
C      00026
C      00027
C      00028
C      00029
C      00030
C      00031
C      00032
C      00033
C      00034
C      00035
C      00036
C      00037
C      00038
C      00039
C      00040
C      00041
C      00042
C      00043
C      00044
C      00045
C      00046
C      00047
C      00048
C      00049
C      00050
C      00051
C      00052
C      00053
C      00054
C      00055
C      00056
C      00057
C      00058
C      00059
C      00060
C      00061
C      00062
C      00063
C      00064
C      00065
C      00066
C      00067
C      00068
C      00069
C      00070
C      00071
C      00072
C      00073
C      00074
C      00075
C      00076
C      00077
C      00078
C      00079
C      00080
C      00081
C      00082
C      00083
C      00084
C      00085
C      00086
C      00087
C      00088
C      00089
C      00090
C      00091
C      00092
C      00093
C      00094
C      00095
C      00096
C      00097
C      00098
C      00099
C      00100
C      00101
C      00102
C      00103
C      00104
C      00105
C      00106
C      00107
C      00108
C      00109
C      00110
C      00111
C      00112
C      00113
C      00114
C      00115
C      00116
C      00117
C      00118
C      00119
C      00120
C      00121
C      00122
C      00123
C      00124
C      00125
C      00126
C      00127
C      00128
C      00129
C      00130
C      00131
C      00132
C      00133
C      00134
C      00135

```


Thesis
S73875 Stone
c.1

199470

A further develop-
ment of the Massachu-
setts Institute of
Technology computer
aided design executive
system.

Thesis
S73875 Stone
c.1

199470

A further develop-
ment of the Massachu-
setts Institute of
Technology computer
aided design executive
system.

thesS73875

A further development of the Massachuset



3 2768 002 02037 2

DUDLEY KNOX LIBRARY